Using ANTLR, implement a short Java program that performs simple expression evaluation. (If

you don’t have ANTLR, download it now. Search for VERSION 3 of ANTLR)The grammar file

is provided for you below. Process the file Expr.g in ANTLR. ANTLR produces three files: a

lexer, a parser and tokens file.

To the grammar add the division operation. It should go with the multiplication operator.

In Netbeans or Eclipse or wherever, create a Java project. Include the three file ANTLR

produced in the project. (ANTLR output can be directed to any directory from ANTLRWorks

FILE-&gt;Preferences-&gt;Output). You my find it convenient to add a package name to the grammar

(for example, package Expression;) in the headers sections for the parser and lexer members. If

not you might manually add them in Netbeans, etc.

Important: ANTLR has a runtime library that must be added to the project. It is called antlr-3.3-

complete (or a different number depending on which you version you have). In Netbeans, expand

the Libraries branch of the project tree, then right click Libraries and selection Add jar, then

navigate to the antlr-3.3-complete location and select it.

At top,

import java.io.IOException;

import org.antlr.runtime.ANTLRFileStream;

import org.antlr.runtime.CommonTokenStream;

import org.antlr.runtime.RecognitionException;

In your Netbeans main place code similar to this:

ExprLexer lexer = new ExprLexer(new ANTLRFileStream(args[0]));

CommonTokenStream tokens = new CommonTokenStream(lexer);

ExprParser parser = new ExprParser(tokens);

parser.prog();

Add the ANTLR runtime library to the project. It is called antlr-x-y-complete.jar where x and y

depend on the version and release numbers for what you downloaded.

In ONE docx, or pdf file, submit your grammar file and Java main class and a screen shout

of the output area for the input given.

Here is the grammar:

grammar Expr;

@header {

package expressionparser;

import java.util.HashMap;

}

@lexer::header {

package expressionparser;

}

@members {

/\*\* Map variable name to Integer object holding value \*/

HashMap memory = new HashMap();

}

prog: stat+ ;

stat: expr NEWLINE {System.out.println($expr.value);}

| ID &#39;=&#39; expr NEWLINE

{memory.put($ID.text, new Integer($expr.value));}

| NEWLINE {System.out.println(&quot;A newline has been issued&quot;);}

;

expr returns [int value]

: e=multExpr {$value = $e.value;}

( &#39;+&#39; e=multExpr {$value += $e.value; }

| &#39;-&#39; e=multExpr {$value -= $e.value;}

)\*

;

multExpr returns [int value]

: e=atom {$value = $e.value;} (&#39;\*&#39; e=atom {$value \*= $e.value;})\*

;

atom returns [int value]

: INT {$value = Integer.parseInt($INT.text);}

| ID

{

Integer v = (Integer)memory.get($ID.text);

if ( v!=null ) $value = v.intValue();

else System.err.println(&quot;undefined variable &quot;+$ID.text);

}

| &#39;(&#39; expr &#39;)&#39; {$value = $expr.value;}

;

ID : (&#39;a&#39;..&#39;z&#39;|&#39;A&#39;..&#39;Z&#39;)+ ;

INT : &#39;0&#39;..&#39;9&#39;+ ;

NEWLINE:&#39;\r&#39;? &#39;\n&#39; ;

WS : (&#39; &#39;|&#39;\t&#39;)+ {skip();} ;

Sample input file contents:

2 + 2

2\*2\*2+5\*7

A=5

B=8

C=A \* B

A

B

C

C-7

16 / 4

55 / 5