**Abstract**  
ANTLR (Another Tool for Language Recognition) is a powerful parser generator for reading, processing, executing, or translating structured text or binary files. It’s widely used to build languages, tools, and frameworks. From a grammar, ANTLR generates a parser that can build parse trees. It has been used for building powerful languages like python. Our primary goal was to develop such grammar that can handle if else statements ,expression ,loop .By using this grammar ,we can solve many kinds of problems. For checking the grammar, we choose two problems. Problem one is factor of a positive integer and another one is Leap Year. In this code, we use loop, if else statement, different expression that we get from the grammar. Last we have generated a parse tree.

**1 Introduction**

Having some working knowledge of compiler and its design is must for a programmer. In our project we have tried to display whatever we have learned about compiler. We have used ANTLR tool to implement our grammar for detecting any kind of code that is related to for, if else statements and expression . We have created our own grammar to complete the task.

**2 Grammar**

grammar prog1 ;

root : declare func ;

declare : '->' 'open' ':' ID ;

func : 'begin' '->' scope ;

statement : (

variable\_declare

| condition\_if

| loop

| print

)+

;

variable\_declare : exp ;

exp : ID SIGN DIGIT | ID SIGN SIGN | ID | ID SIGN | ID SIGN ID | ID SIGN DIGIT SIGN DIGIT | ID SIGN ID SIGN ID ;

condition\_if: 'if' '(' check ')' scope | 'if' '(' check ')' scope 'eif' '(' check ')' scope 'e' scope | 'if' '(' check ')' scope 'e' scope | 'e' scope;

loop:'for:' 'from' exp 'to' exp scope | 'while:' exp scope ;

print : 'out<' ID ;

check : ID SIGN DIGIT | ID SIGN SIGN | exp SIGN DIGIT ;

SIGN : '=' | '+' | '-' | '\*' | '/' | '%' | '<=' | '>=' | '!=' | '==' | '++' | '--';

ID : [a-zA-Z]+ ;

DIGIT: [0-9]+ ;

scope : '{' statement '}' ;

WS : [ \t\r\n]+ -> skip ;

**3 Demo Code 1**

-> open : stdmode

begin ->

{

for: from i = 1 to i <= 100

{

ceven = 0

codd = 0

for: from j = 1 to j <= i

{

if ( i % j == 0)

{

out< j

}

e

{

if ( j % 2 == 0 )

{

ceven = ceven + 1

}

e

{

codd = codd + 1

}

}

j = j + 1

}

out< i

out< ceven

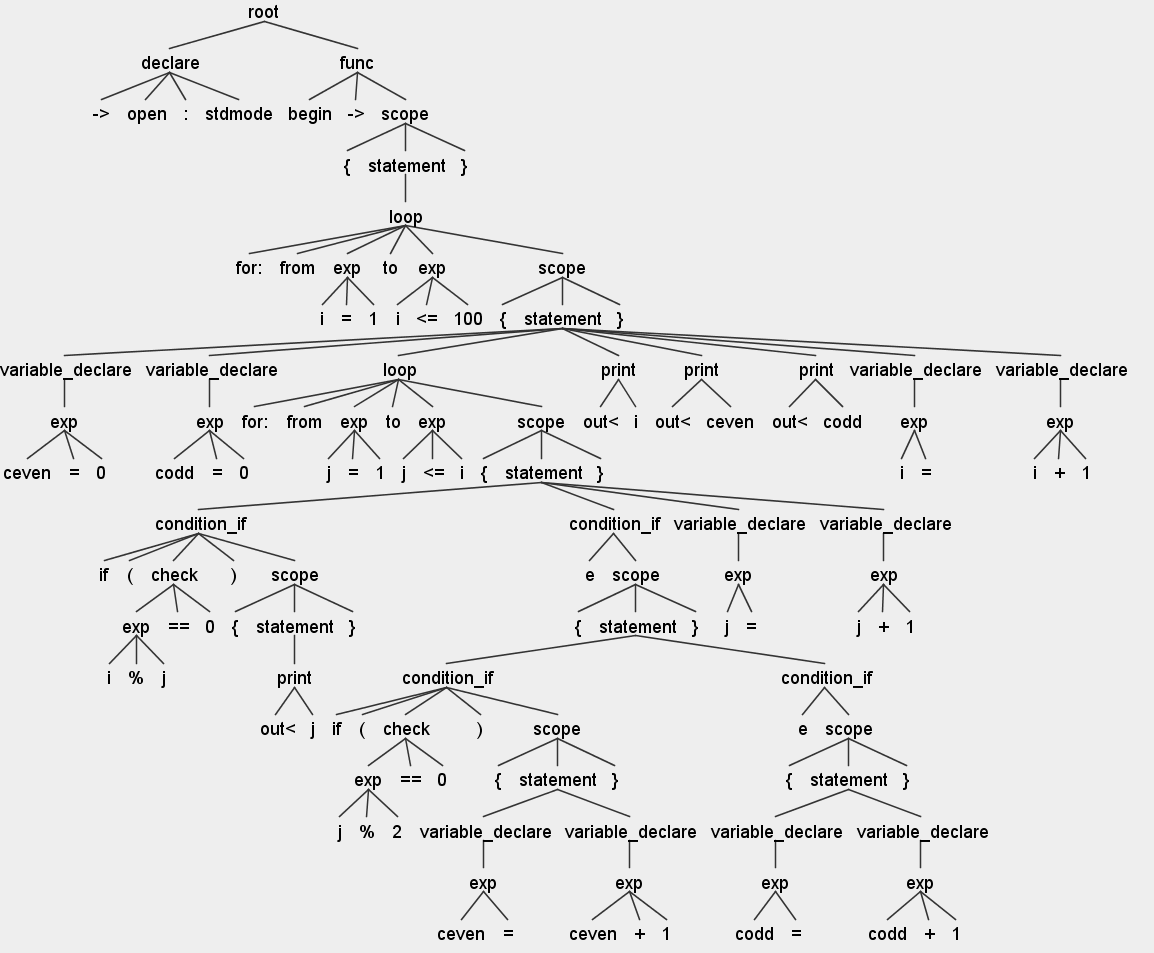
out< codd

i = i + 1

}

}

**4 Demo Code 1 Parse Tree**

****

**5 Demo Code 2**

-> open : stdmode

begin ->

{

for:from year = 1900 to year <= 2100

{

if ( year % 4 == 0 )

{

if( year % 100 == 0 )

{

if ( year % 400 == 0 )

{

out< Leap year

}

e

{

out< Not Leap year

}

}

e

{

out< Not Leap year

}

}

e

{

out< Not Leap year

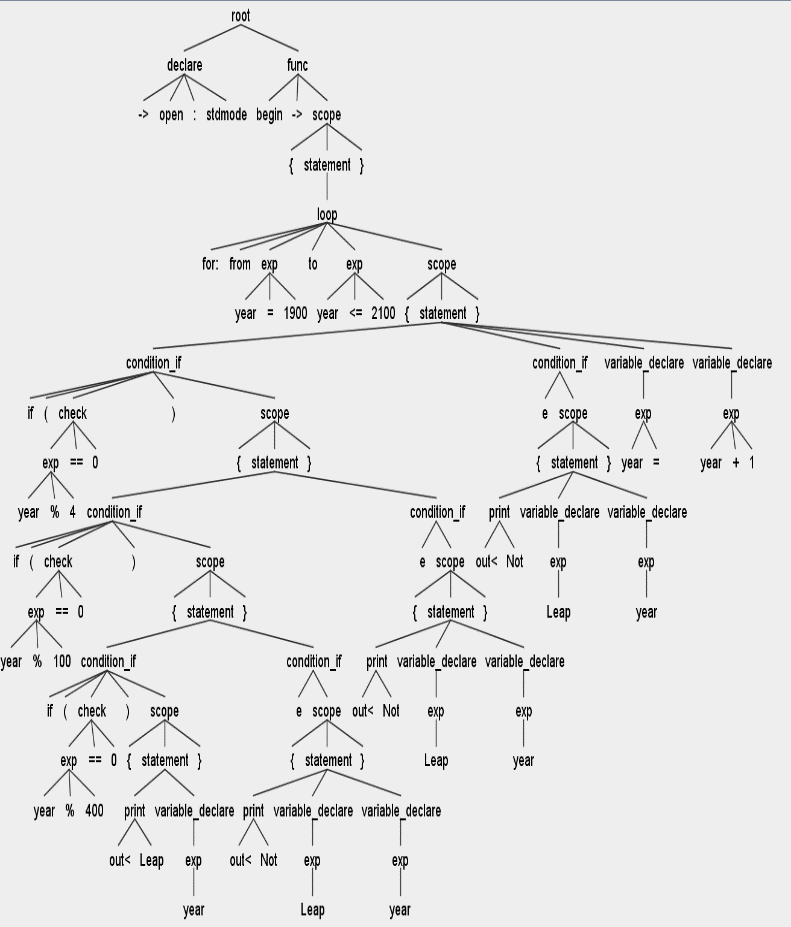
}

year = year + 1

}

}

**6 Demo Code 2 Parse Tree**

****

**7 Conclusion**

The rule of the grammar is so easy that anyone can write a program very easily. Most the problems that arises can be solved using if else statements ,loop ,expression. It means we can solve most the problem. We have design the grammar in such a way that we can extend the grammar in future and add more header files.