Compiler Design on Programming Language

MD. Shanewaz Akib (ID: 2016-3-60-050)

md\_shanewaz@yahoo.com

Shahariar Bhuiyan (ID: 2015-0-60-000)

sr.raghib@gmail.com

*Department of Computer Science &Engineering*

*East West University,Dhaka*

***Abstract***— ***Our project is about creating parse tree and separating tokens using ANTLR tool. We have mainly focused on how the tool ANTLR is working and by using it creating a parse tree or abstract syntax tree from the written grammar. For using this tool, we also need JDK (Java Development Kit). In our project, we have tried to generate a parse tree based on the programming language for find prime number. Our grammar create parse tree to find prime number from 0 to 100.In this code, we use for loop, nested for loop if condition, different statement and so on that we get from the grammar. Finally we generate a parse tree which determine prime number from 0 to 100***

***Keywords***— ***ANTLR, parse tree, abstract syntax tree, JDK,***

***prime number, for loop, nested for loop, if condition.***

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. In our project, a parse tree is generated and the tokens are being separated.

**II. PARSING A FUNCTION**

1. Understanding the structure of the function

2. Converting function structure according our regular expression

3. Including the default words as terminal symbols

4. Defining the type of function

**III. IDENTIFYING TOKENS AND GENERATING CLASS FILES**

1.Identifying tokens by ANTLR

2. Generating JAVA Class Files

3. Running the JAVA Files

4. Generating Parse Tree Using GUI

**IV. PROPOSED WORK**

1. The grammar program file (.g4) is executed using the ANTLR tool and the generated tokens files are run with java and it generates the class files.
2. Java compiler parses source code of .**java** file and creates tokens that match the **java** grammar.
3. After executing the java file an input is required. After the input the java program generates a parse tree and shows that tree using Graphical User Interface (GUI).

**V. USING COMMAND LINE WHICH RUN JKD AND ANTLR-4.7.2-COMPLETE.JAR TO GENERATE PARSE TREE**

antlr program.g4

javac program\*.java

class

input program start input.txt

**VI. GRAMMAR (.g4 FILE)**

grammar program ;

start: declaration func;

declaration: '@' declist dectype ';' ;

declist : 'Include' | 'Define' ;

dectype : term '\_' term ;

term : ID | Digit | ID op Digit;

func: term term '[' ']' parenmain ;

parenmain: 'beginmain' statement 'endmain';

statement: (

variable\_declare

| loop

| if\_cond

| countstat

| breakstat

| display

| returntype

)+

;

variable\_declare: declare '.';

declare: term relop term | term term;

var: ID ;

incr\_decr : '++'| '--' ;

op : 'plus' | 'minus' | 'multiply' | 'devide' | 'mod' ;

relop : 'equal to' | 'not equal' | 'greater than equal' | 'less than' | 'greater than' | 'less than equal' | '=' | '==' ;

loop: loopstmt ;

loopstmt: 'for''['cond']' parenfor ;

cond: var'='term ';' var relop term ';' var incr\_decr | term op term relop term | term relop term ;

parenfor: 'beginfor' statement 'endfor' ;

parenif: 'beginif' statement 'endif' ;

parenelif: 'beginelif' statement 'endelif' ;

if\_cond: 'if' '[' cond ']' parenif | 'if' '[' cond ']' parenif 'elif' parenelif ;

countstat: term incr\_decr '.' ;

breakstat: 'break' '.';

display: 'print' '[' term ']' '.' ;

returntype: 'return' term '.';

ID : [a-zA-Z]+ ;

Digit : [0-9]+ ;

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

**VII. CODE (INPUT FILE)**

@Include stdio\_h;

int start[]

beginmain

int num.

int count.

for[num=1;num less than equal 100;num++]

beginfor

count equal to 0.

for[k=2;k less than equal num devide 2;k++]

beginfor

if[num mod k == 0]

beginif

count++.

break.

endif

endfor

if[count == 0]

beginif

print[num].

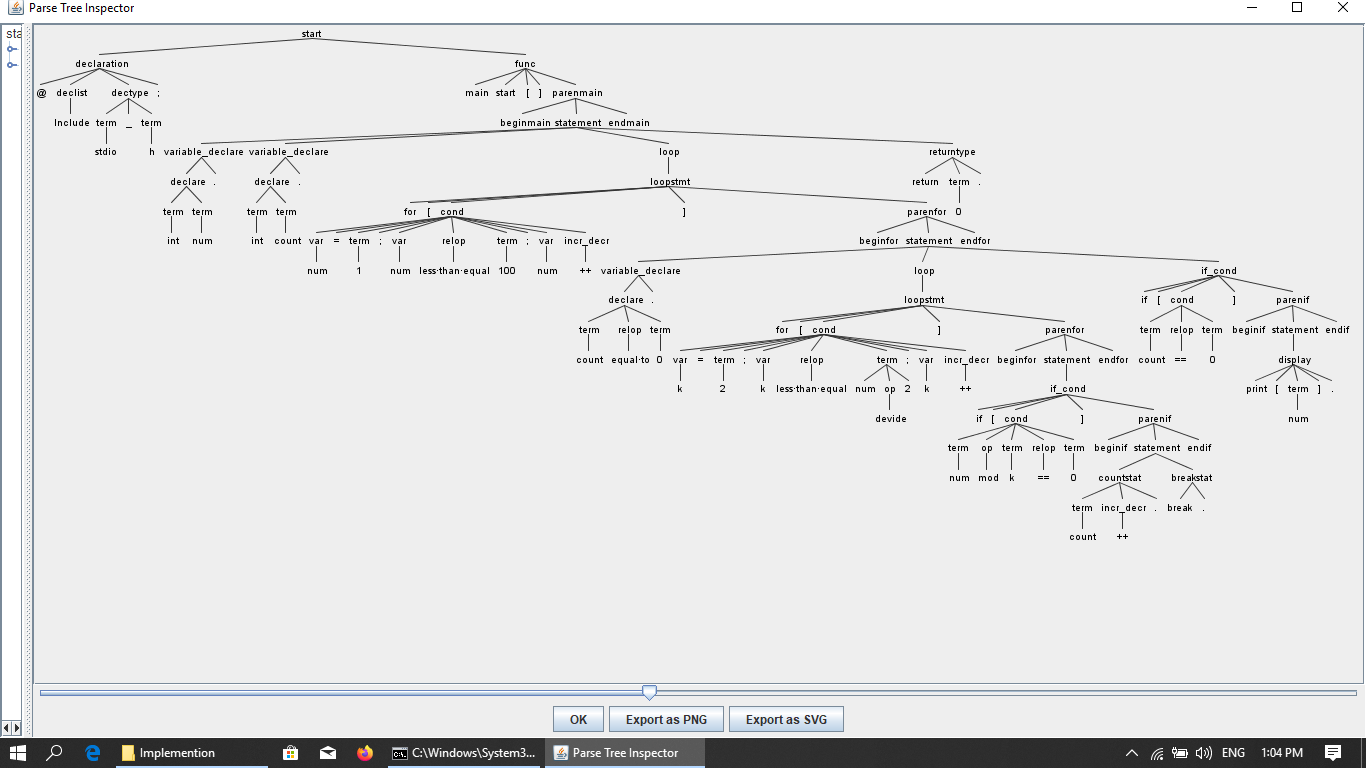
endif

endfor

return 0.

endmain

**VIII. PARSE TREE**

****

**XI. CONCLUSION**

Our project will help to find prime number from

0 to 100 by creating a parse tree. 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, nested loop,

expression. we also solve most this problem.

We have parsed a function in such a way that

who is new to programming she/he can

understand easily function type, variable,

conditions, if-else, for loop, nested for loop

etc. We have design the grammar in such a way

that we can extend the grammar in future and

add more header files.

**X. REFERENCES**

1.Abstract syntax tree

<https://en.wikipedia.org/wiki/Abstract_syntax_tree>

2.ANTLR

<https://www.antlr.org/>

#### 3.Java SE - Downloads | Oracle Technology Network | Oracle

#### <https://www.oracle.com/technetwork/java/javase/downloads/index.html>

#### 4.C program to display the prime numbers between 1 and 100

#### <http://c.net-question.in/c-basic-programs/prime-numbers-between-1-and-100>