****

**CSE 375: Compiler Design**

**Section: 01, Fall-2020**

|  |
| --- |
| A Project Report  on  **Implementation of Parse Tree and Token from a Grammar Using Antlr** |
| Submitted by:  Group Members:  NAME ID  SHEKHOR CHANDRA SAHA 2017-3-60-025 |
| Submitted to:  Dr. Shamim H Ripon  Professor, Dept. of CSE |
| Date of Submission: 07-01-2021 |

**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

**EAST WEST UNIVERSITY**

**Description:**

Compiler is a program takes a program written in a source language and translate it into an equivalent program in a target language. The process of compilation carried out in two parts: analysis and synthesis.A parse tree represents the structure of the derivation of a compiler. First step of a compiler is to create a parse tree of the program tree. Compiler design principles provide an in-depth view of translation and optimization process. Compiler design covers basic translation mechanism and error detection & recovery. It includes lexical, syntax, and semantic analysis as front end, and code generation and optimization as back-end.

A context-free grammar is a set of recursive rules used to generate patterns of strings. A context-free grammar can describe all regular languages and more, but they cannot describe all possible languages.

Example

The grammar {\G=(\{S\},\{a,b\},P,S)} G=(\{S\},\{a,b\},P,S), with productions

S → aSa,

S → bSb,

S → ε

**Context free Grammar:**

grammar ggg ;

root : declaration declarative function ;

declaration : '!' declarationlist '('declarationtype ')' ;

declarationlist : 'include' | 'define' ;

declarationtype :

'iostream' | 'stdio.h' | 'conio.h' | 'math.h' | 'string.h' | term '.' term ;

declarative : 'using namespace std ;' ;

function : 'int' ID '[' ']' block ;

block : '{' statement '}' ;

statement :(

expression\_statement

| selection\_statement

| iteration\_statement

| return\_statement

| compound\_statement

| output\_statement

| break\_statement

)+

;

expression\_statement : typeSpecifier expr ';' | ';' ;

return\_statement : 'back' expr ';' | 'back' term ';' ;

expr : expr binop expr | '(' expr ')' | term ;

binop : 'plus' | 'minus' | 'multiply' | 'divide' | relop ;

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

logical\_op : 'AND' | 'OR' ;

selection\_statement : 'if' '[' expr ']' block | 'if' '[' expr ']' block 'if other' block ;

compound\_statement : '{' expr '}' ;

break\_statement : 'break' ;

iteration\_statement : conditional\_statement | loop\_statement ;

conditional\_statement : 'condition' '[' expr ']' block ;

loop\_statement : 'repeat''['loopexpr']' block ;

loopexpr : var '='term '/' var relop term '/' var incr\_op ;

output\_statement : 'printout' '<<' expr ';' ;

var : ID ;

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

term : ID | LIT ;

typeSpecifier : 'integer' | 'boolean' | 'character' ;

ID : [a-zA-Z]+ ;

LIT : [0-9]+ ;

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

**The grammar I used:**

Declaration :

!include(iostream)

Selection statement :

'if' '[' expr ']' block | 'if' '[' expr ']' block 'if other' block

Conditional statement :

'condition' '[' expr ']' block

Loop statement :

'repeat''['loopexpr']' block

loopexpr :

var '='term '/' var relop term '/' var incr\_op

Output statement :

'printout' '<<' expr ';'

Return statement :

'back' expr ';' | 'back' term ';'

**Actual C++ grammar:**

Declaration:

#include<iostream>

Selection statement :

'if' '(' expr ')' block | 'if' '(' expr ')' block 'else' block

Conditional statement :

'while' '(' expr ')' block

Loop statement :

'for''(' loopexpr ')' block

loopexpr :

Var '='term ';' var relop term ';' var incr\_op

Output statement :

'cout' '<<' expr ';'

Return statement :

'return' expr ';' | 'return' term ';'

**Correct Input:**

!include (iostream)

using namespace std ;

int func[]

{

integer a=10;

integer b=7;

integer s=0;

condition[s less than a]

{

if[s less than 100]

{

integer i;

repeat[i=0 / i less than 5 / i ++]

{

back s=s plus a;

}

printout << s;

integer number=10;

integer n=5;

integer j = 1;

integer factorial = 1;

integer count = 0;

condition[count less than n]

{

repeat[j = 1 / j less than equal number / j ++ ]

{

back factorial = factorial multiply j ;

}

printout << factorial;

back count = count plus 1;

}

}

if other

{

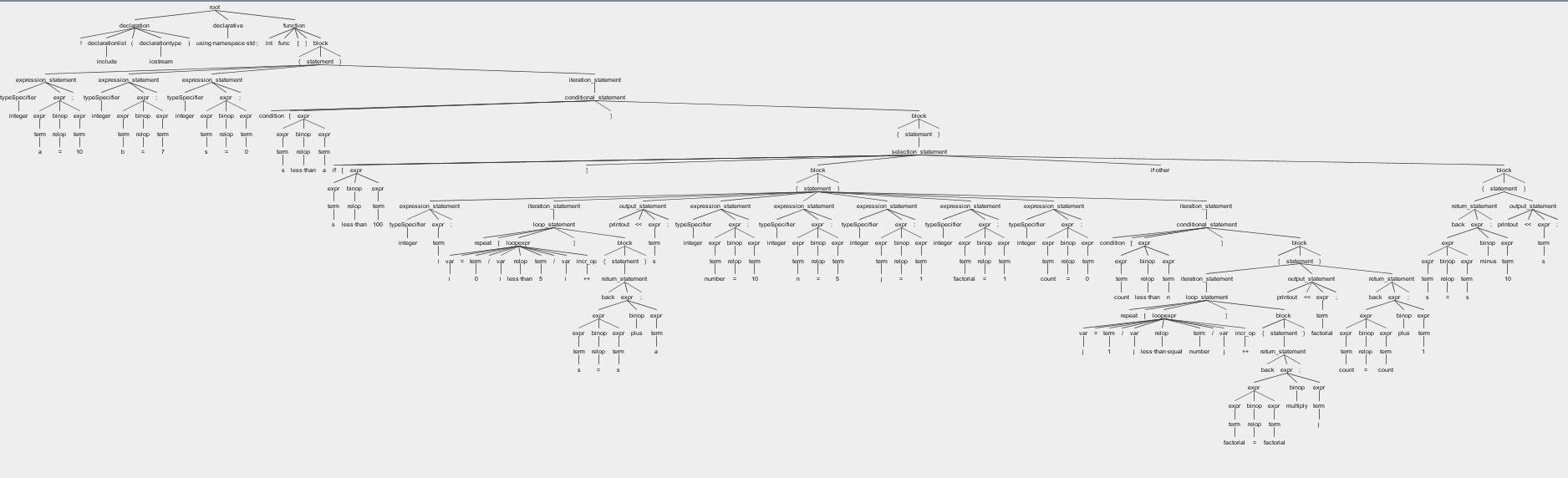
back s=s minus 10;

printout << s;

}

}

}

**Correct Input Parse Tree:**

**Wrong Input:**

@include <iostream>

integer func{}

[

int a=10;

int b=7;

int s=0;

loop{s less than a}

[

if{s less than 100}

[

int i;

for{i=0 / i less than 5 / i =i+2}

[

back s=s plus a;

]

print " s ";

int number=10;

int n=5;

int j = 1;

int factorial = 1;

int count = 0;

loop{count less than n}

[

for{j = 1 / j less than equal number / j ++ }

[

return factorial = factorial multiply j ;

]

print " factorial ";

return count = count plus 1;

]

]

if other

[

return s=s minus 10;

print " s ";

]

]

]

**Wrong Input Parse Tree:**

