East West University

Project

Course Title	: Compiler Design
Course Code	: CSE-375
Section	: 03
Project Title	: Generating a Parser for a customized (imaginery) programming language.

Submitted To

Dr. Shamim H Ripon
Professor
Department Of CSE
East West University

Group members:

Zubayar Mahatab Md Sakif	**2018-1-60-105**
Sanjida Reza Rafa	**2017-1-60-004**
Maisha Maliha	**2018-1-60-111**
Md. Mokibul Hasan Antor	**2017-3-60-019**

Date of Submission: 28-05-2021

Grammar Overview:

- Variable type: 'integer'|'double'|'boolian'|'char';
- Conditional statement(if..elseif…else)
- print statement
- relational operator (=,!=,>,<,>=,=)
- Input statement: (scan_)
- Loop statement: For loop and while loop, nested for/while loop
- Break statement : using break in a loop (‘endloop’)
- Output statement: (print)
- ID : [a-zA-Z]+ ;
- LIT : [0-9]+ ;

Structure of the code

Header Part:

```
^^include<stdio.h>

^^include<iostream>

^^define MAX_SIZE 1000

^^import<stdio.h>

^^import<iostream>
```

Function

Type(‘integer’/’double’/’boolian’/’char’)

```
(ID/Type ID) function_name(Parameter1, parameter2,…)
^<
    Inner_part;
>^
```

Main Function

```
Integer main()
^<
        Inner_part;
>^
```

Function Call

```
Function_name(Parameter1, Parameter2,…);

Function_name();
```

Variable Declaration

```
Type variable_name ;
```

Type: ('integer'/'double'/'boolian'/'char')

Variable Implementation

Type variable_name = ID/LIT ;

Type: ('integer'/'double'/'boolian'/'char')

ID : (a-z,A-Z) ;

LIT : (0-9) ;

Variable assign:

Variable= ID/LIT;

ID : (a-z,A-Z) ;

LIT : (0-9) ;

Array :

Type array_name[ID/LIT];

Input:

scan ^< $Type : $Variable_name >^;

Output:

print ^<$%integer is a prime number : x>^

Conditional statement

```
If: if^<any condition>^
^<
    Inner_part;
>^
Else: else
^<
        Inner_part;
  >^
```

For loop

```
for^<initialize ;condition; increment/decrement>^
^<
        Inner_part;
 >^
```

Binary_operations : ('$+' / '$-' / '$*' / '$/' / '$%')

Relational_operation: ('$=' / '$!=' / '$>' / '$>=' / '$<' / '$<='/'$==')

Variable_increment_decrement: ('$++'/ '$--')

While loop

```
while^<conditiont>^
^<
        Inner_part;
 >^
```

Switch_case

```
switch(argument)
^<
    case 0:
    ^<
           inner_part;
           break;
     >^
     case 1:
     ^<
           inner_part
           break;
     >^
```

>^

Grammar

```
grammar project;

root: declaration function+ ;

declaration:('^^' declarationlist ('<' declarationtype '>'|declarationtype))+ ;

declarationlist : 'include' | 'define' | 'import' ;

declarationtype: term '.' term| expression+;


function :((ID|type ID) '(' ')' inner_part) |( (ID|type ID) '(' type variable ')' (';')?(inner_part)? )
|( (ID|type ID) '(' (type variable ',' type variable)+ ')' (';')? (inner_part)? ) ;

inner_part: '^<' information '>^';

information:

(

about_expr

| if_else

| return_

| iteration

| output

| breakset

| scan_

| functioncall

| switch_case

)+

;

about_expr: (type term+ ((','term+)+)?) ';'|(type)? term'$=' term('['term']')?(','(variable|term'$='
term))?';'|(type)? (term+ '[' term ('_'term)? ']'(',')?)+(','(variable|term'$='
term))?';'|term+'['term']' rel_op (term+'['term']'|symbol term symbol) ';'|term variable_inc_dec
';'|(type)? term+ rel_op functioncall ';'| (type)?term+ rel_op term bin_op term ';' ;

return_: 'return' expression ';' | 'return' term ';'| 'return' (expression+)? functioncall ';' ;
```

```
expression :symbol+|term+|expression bin_op expression |expression rel_op expression
|expression logic_op expression|term (term',')+ term | expression rel_op term|term
'['term']'rel_op term|term bin_op term rel_op term|term+('['term']')? rel_op (symbol)? term
(symbol)?;

symbol: '*' | '@' | '!' |'-' |'_' |'~'| '/'|'?'|';'|'""'| ','| '.'|':';

bin_op:'$+' | '$-' | '$*' | '$/' | '$%';

rel_op:'$=' | '$!=' | '$>' | '$>=' | '$<' | '$<='|'$==';

logic_op:  '$||' | '$&&' ;

if_else: 'if' '^<' expression ((logic_op expression+)+)? '>^'inner_part | 'if' '^<' expression
((logic_op expression+)+)? '>^' inner_part 'else' inner_part| 'if' '^<' expression ((logic_op
expression+)+)? '>^' inner_part 'else if' '^<' expression ((logic_op expression+)+)? '>^'
inner_part | 'if' '^<' expression ((logic_op expression+)+)?'>^' inner_part 'else if'
'^<'expression ((logic_op expression+)+)?'>^' inner_part 'else' inner_part ;

breakset: 'break'';' | 'continue' ';' ;

switch_case : 'switch' '(' expression+ ')' '^<' switchblock '>^' ;

switchblock : ('case' term ':' inner_part )+ ('default' ':' inner_part)?;

iteration: condition | loop;

condition: 'while' '^<' expression+'>^' inner_part;

loop: 'for' '^<' (type)? variable '$=' term ';' variable rel_op term ';' (variable
variable_inc_dec|variable_inc_dec variable) '>^' inner_part;

output: 'print' '^<' expression ':' '>^' ';'| 'print' '^<' bin_op type (expression)? ':'
variable('['variable']')? '>^' ';'|'print' '^<' expression+ '>^' ';'|'print' '^<' expression+ (rel_op)?
bin_op type (expression+)? (rel_op)? (bin_op type)? ':' expression+ (functioncall)? '>^' ';' |
'print' '^<' (expression bin_op type)+ ':' expression '>^'';'|'print' '^<' expression bin_op type
term+ bin_op type ':' expression functioncall '>^' ';'|'print' '^<'bin_op type expression+ ':'
expression+ '>^'';';

scan_: 'scan'  '^<' (bin_op type)+ ':' ('$'term+)+ ('['variable']')?'>^' ';'|'gets''^<'term+'>^'';';

functioncall: variable '(' ')'(';')?|  variable '(' (expression+)? ')'(';')? ;

variable: ID;

variable_inc_dec:'$++'| '$--';

term:ID|LIT ;

type: 'integer'|'double'|'boolian'|'char';
```

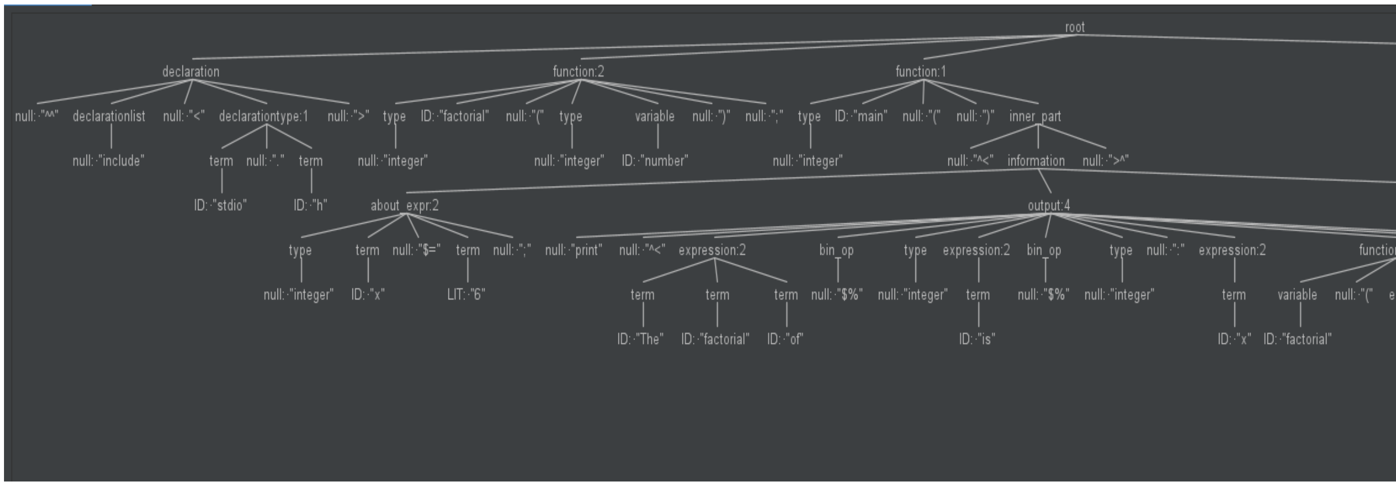
```
ID : [a-zA-Z]+ ;

LIT : [0-9]+ ;


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

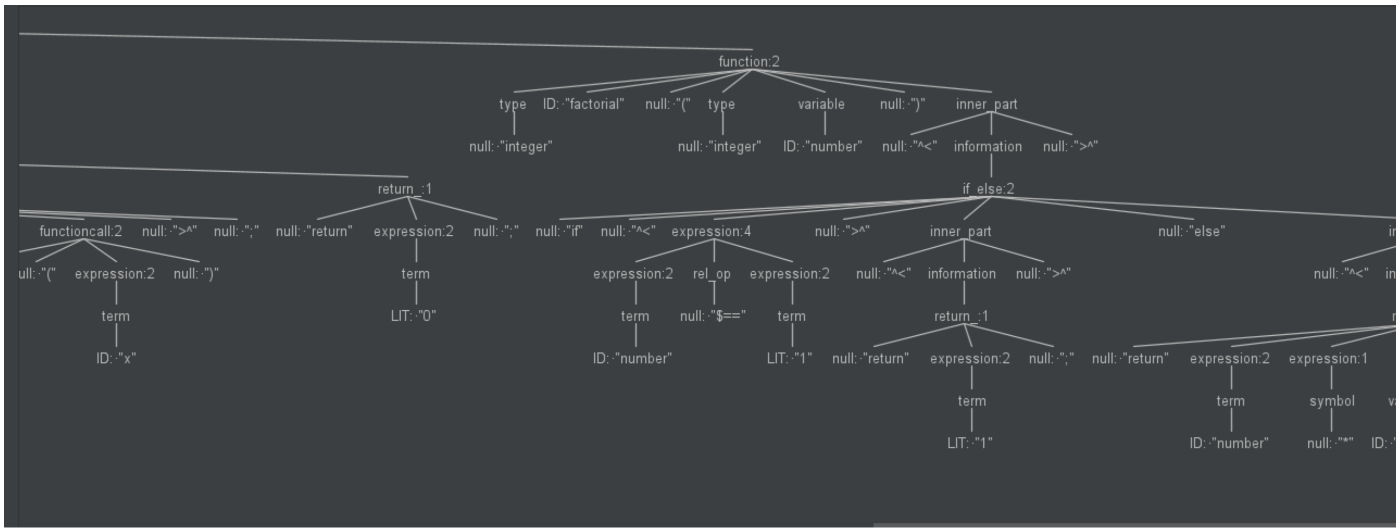
Sample correct input:

1. Function-recursion

```
 ^^include <stdio.h>

integer factorial(integer number);
integer main()
^<
integer x $= 6;
print ^<The factorial of $%integer is $%integer: x factorial(x)>^;
return 0;
>^
integer factorial(integer number)
^<
if ^< number $== 1>^
^<
return 1;
>^
else
^<
return number * factorial(number - 1);
>^
>^
```

Tree



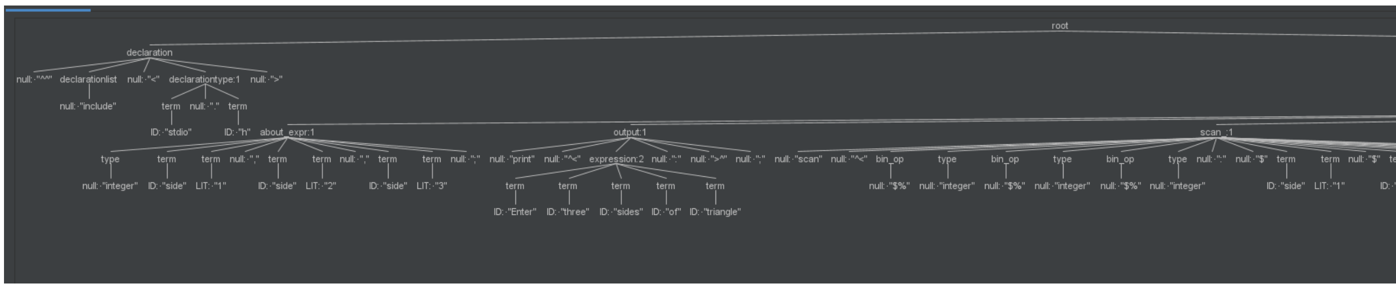


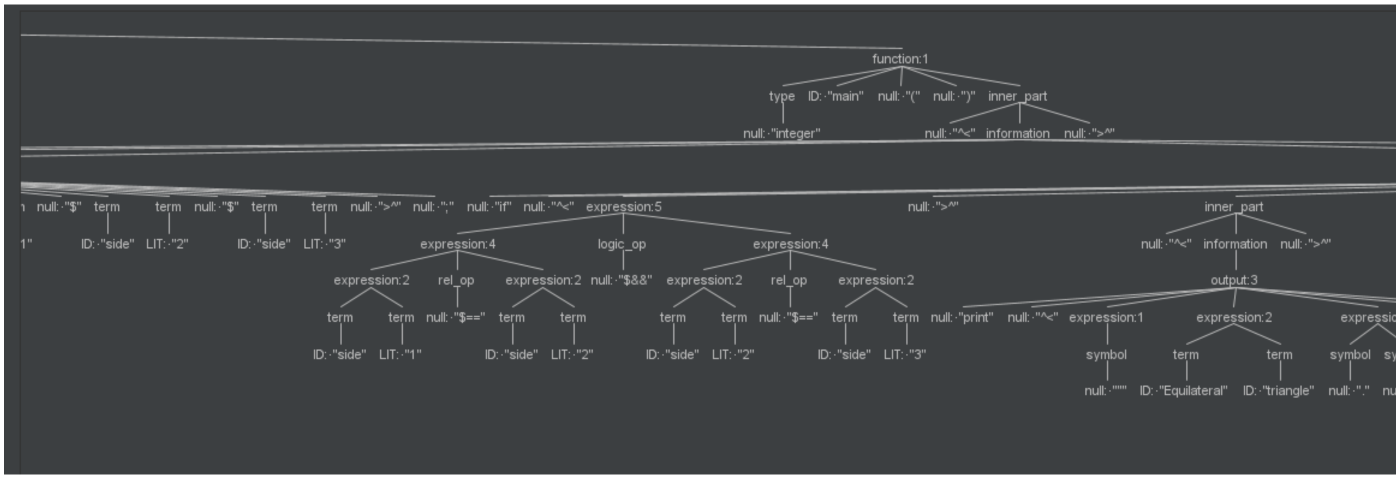
2. if-else

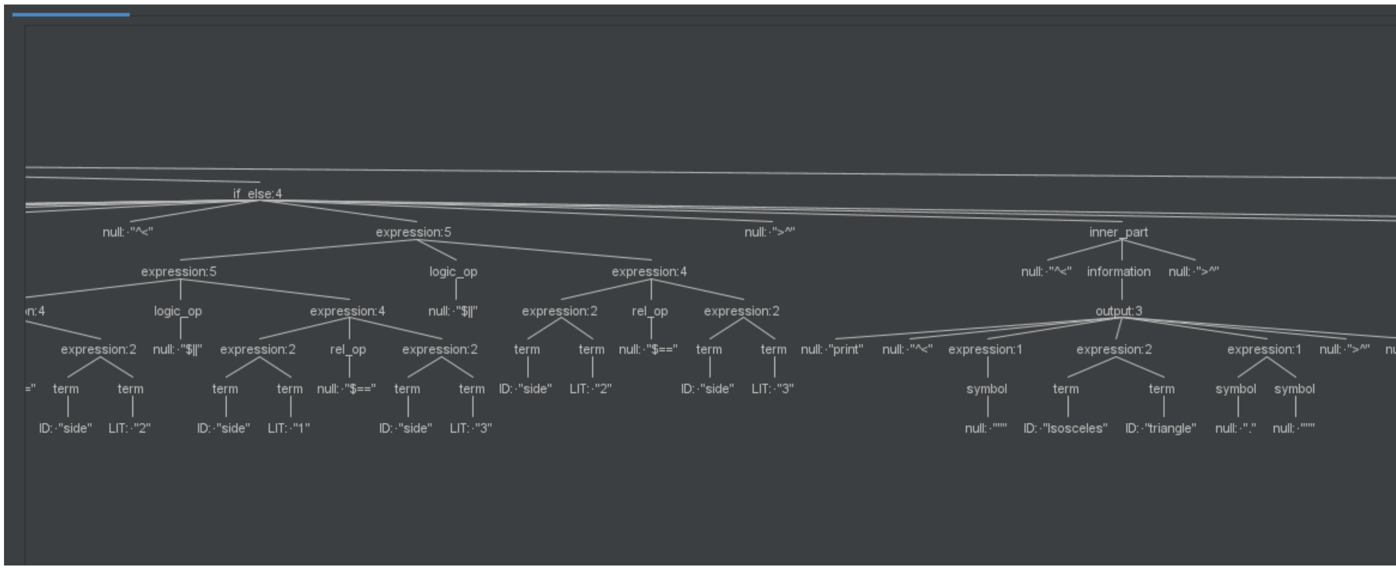
```
 ^^include <stdio.h>

integer main()
^<
integer side1, side2, side3;
print^<Enter three sides of triangle: >^;
scan^<$%integer$%integer$%integer: $side1 $side2 $side3>^;
if^<side1 $== side2 $&& side2 $== side3>^
^<
print^<"Equilateral triangle.">^;
>^
else if^<side1$==side2 $|| side1$==side3 $|| side2$==side3>^
^<
```

```
print^<"Isosceles triangle.">^;
>^
else
^<
print^<"Scalene triangle.">^;
>^
return 0;
>^
```

Tree:


root
declaration
null: "^^" declarationlist null: "<" declarationtype:1 null: ">"
null: "include"
term null: "." term
ID: "stdio" ID: "h"
about_expr:1
type term term null: "," term term null: "," term term null: ";"
null: "integer" ID: "side" LIT: "1" ID: "side" LIT: "2" ID: "side" LIT: "3"
output:1
null: "print" null: "^<" expression:2 null: ":" null: ">^" null: ";"
term term term term term
ID: "Enter" ID: "three" ID: "sides" ID: "of" ID: "triangle"
scan_:1
null: "scan" null: "^<" bin_op type bin_op type bin_op type null: ":" null: "$" term term null: "$"
null: "$%" null: "integer" null: "$%" null: "integer" null: "$%" null: "integer"
ID: "side" LIT: "1"

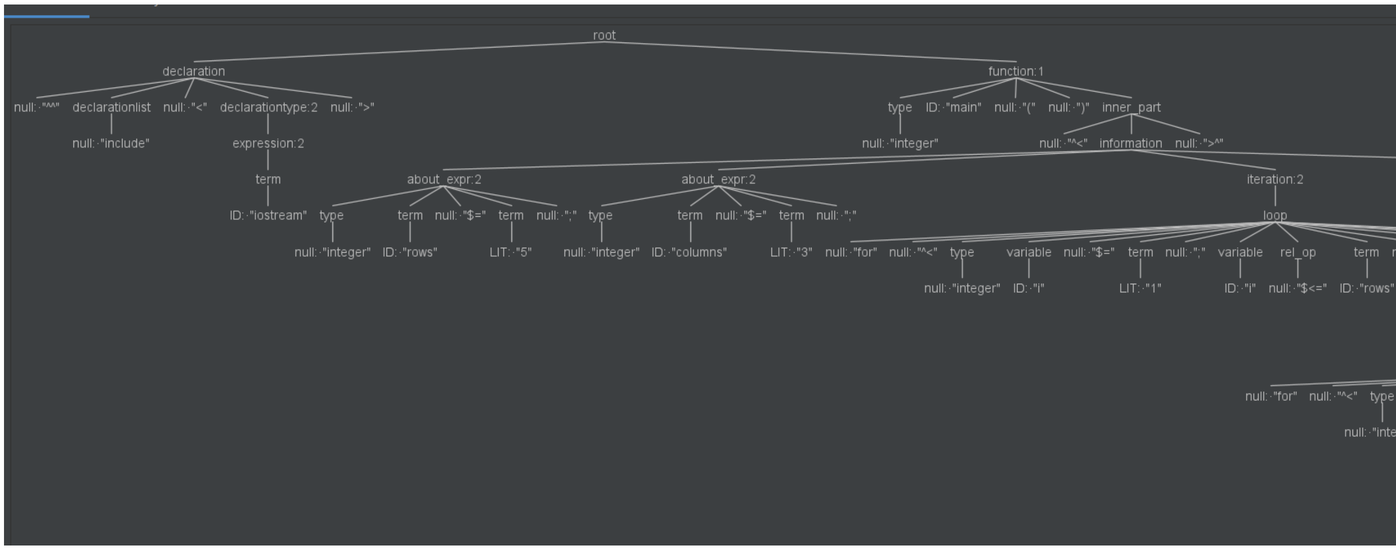

function:1
type ID: "main" null: "(" null: ")" inner_part
null: "integer"
null: "^<" information null: ">^"
null: "$" term term null: "$" term term null: ">^" null: ";" null: "if" null: "^<" expression:5
1" ID: "side" LIT: "2" ID: "side" LIT: "3"
expression:4 logic_op expression:4
expression:2 rel_op expression:2 null: "$&&" expression:2 rel_op expression:2
term term null: "$==" term term term term null: "$==" term term
ID: "side" LIT: "1" ID: "side" LIT: "2" ID: "side" LIT: "2" ID: "side" LIT: "3"
null: ">^"
inner_part
null: "^<" information null: ">^"
output:3
null: "print" null: "^<" expression:1 expression:2
symbol term term symbol
null: """ ID: "Equilateral" ID: "triangle" null: "."

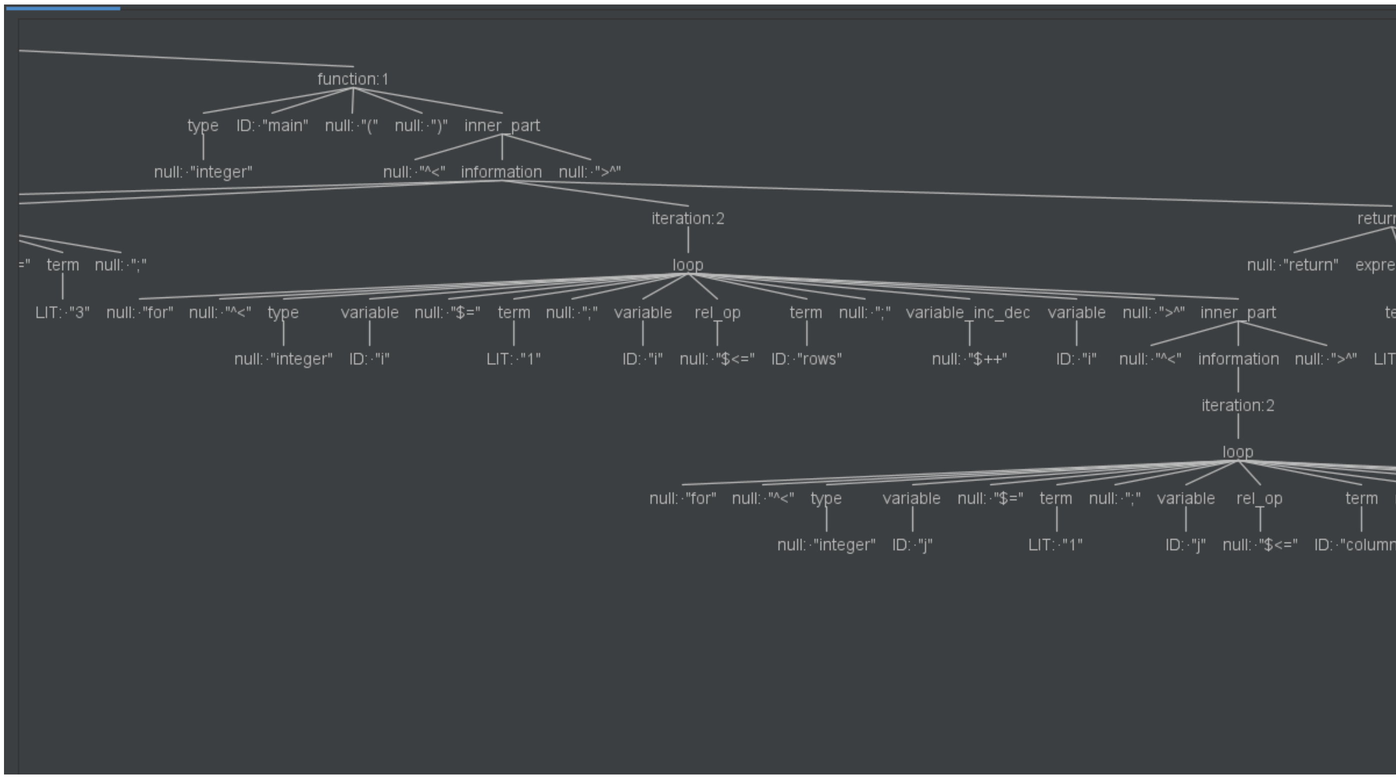


3.Nested-for loop

```
 ^^include <iostream>

integer main ()
^<
integer rows $= 5;
integer columns $= 3;
for ^<integer i $= 1; i $<= rows; $++i >^
^<
for ^<integer j $= 1; j $<= columns; $++j>^
^<
print ^<a b>^;
>^
>^
return 0;
>^
```

Tree

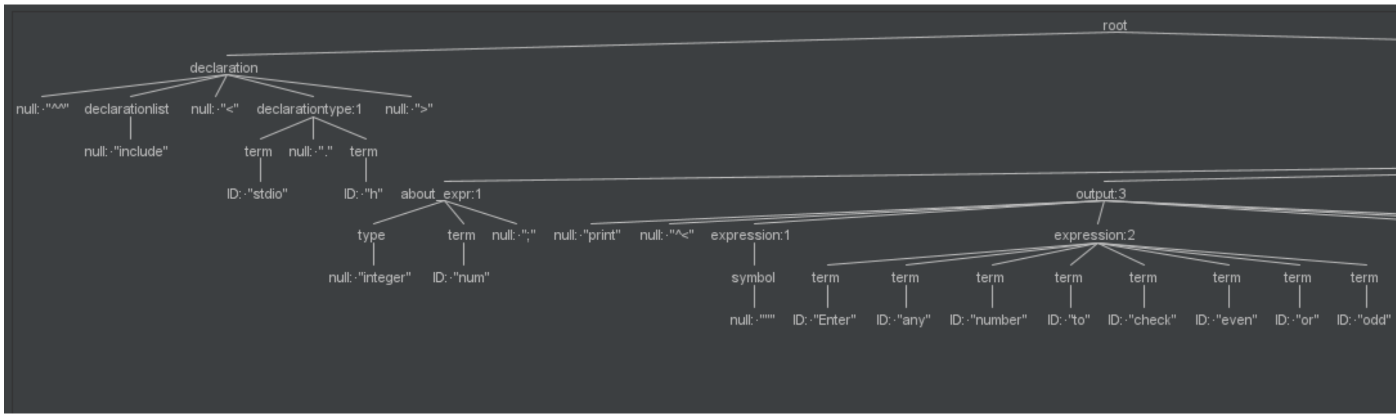

root
declaration
function:1
null: "^^"
declarationlist
null: "<"
declarationtype:2
null: ">"
type
ID: "main"
null: "("
null: ")"
inner_part
null: "include"
expression:2
null: "integer"
null: "^<"
information
null: ">^"
term
about_expr:2
about_expr:2
iteration:2
ID: "iostream"
type
term
null: "$="
term
null: ";"
type
term
null: "$="
term
null: ";"
loop
null: "integer"
ID: "rows"
LIT: "5"
null: "integer"
ID: "columns"
LIT: "3"
null: "for"
null: "^<"
type
variable
null: "$="
term
null: ";"
variable
rel_op
term
null: "integer"
ID: "i"
LIT: "1"
ID: "i"
null: "$<="
ID: "rows"
null: "for"
null: "^<"
type

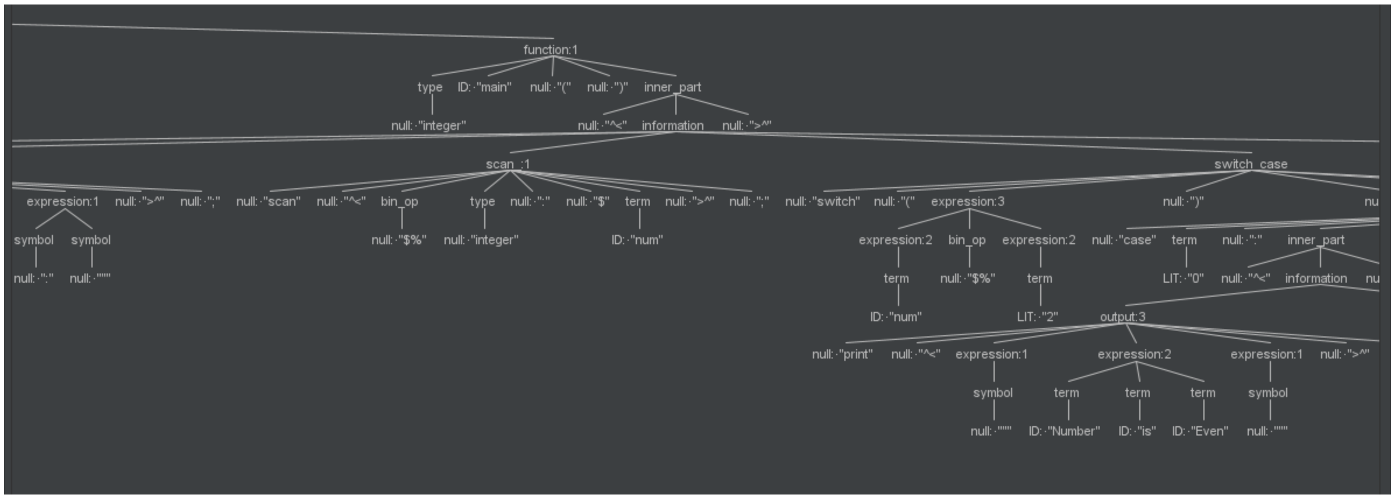

function:1
type
ID: "main"
null: "("
null: ")"
inner_part
null: "integer"
null: "^<"
information
null: ">^"
iteration:2
term
null: ";"
loop
null: "return"
LIT: "3"
null: "for"
null: "^<"
type
variable
null: "$="
term
null: ";"
variable
rel_op
term
null: ";"
variable_inc_dec
variable
null: ">^"
inner_part
null: "integer"
ID: "i"
LIT: "1"
ID: "i"
null: "$<="
ID: "rows"
null: "$++"
ID: "i"
null: "^<"
information
null: ">^"
iteration:2
loop
null: "for"
null: "^<"
type
variable
null: "$="
term
null: ";"
variable
rel_op
term
null: "integer"
ID: "j"
LIT: "1"
ID: "j"
null: "$<="

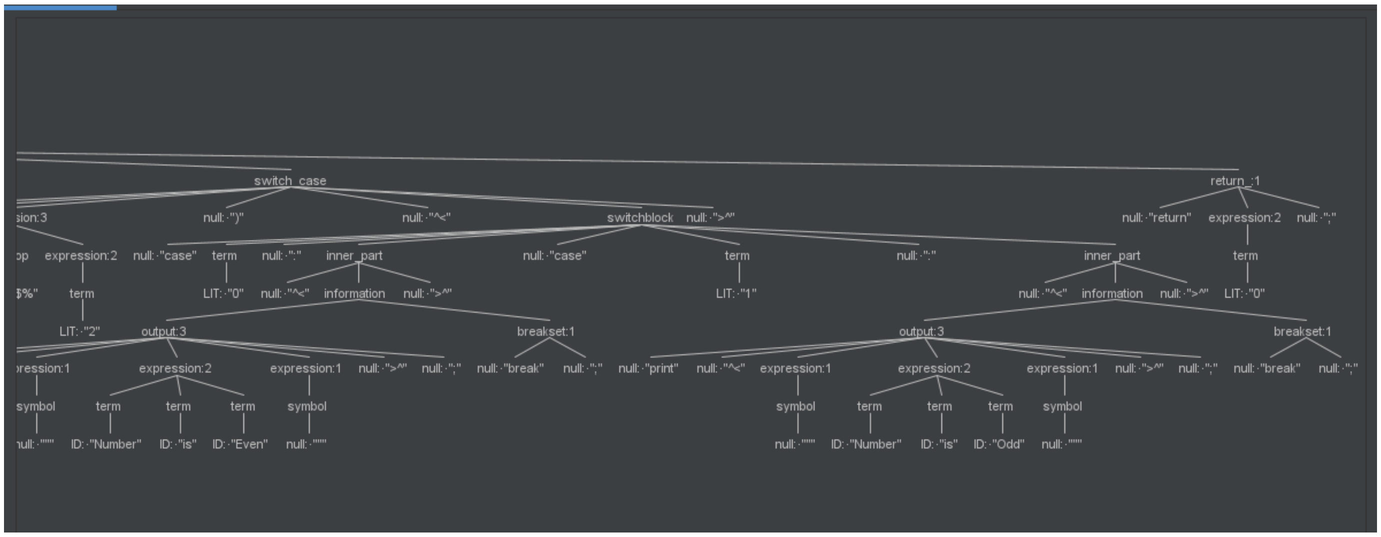
4.Switch-case

```
^^include <stdio.h>

integer main()
^<
integer num;
print^<"Enter any number to check even or odd: ">^;
scan^<$%integer: $num>^;
switch(num $% 2)
^<
case 0:
^<
print^<"Number is Even">^;
break;
>^
case 1:
^<
printf("Number is Odd");
break;
>^
>^
return 0;
>^
```

Tree





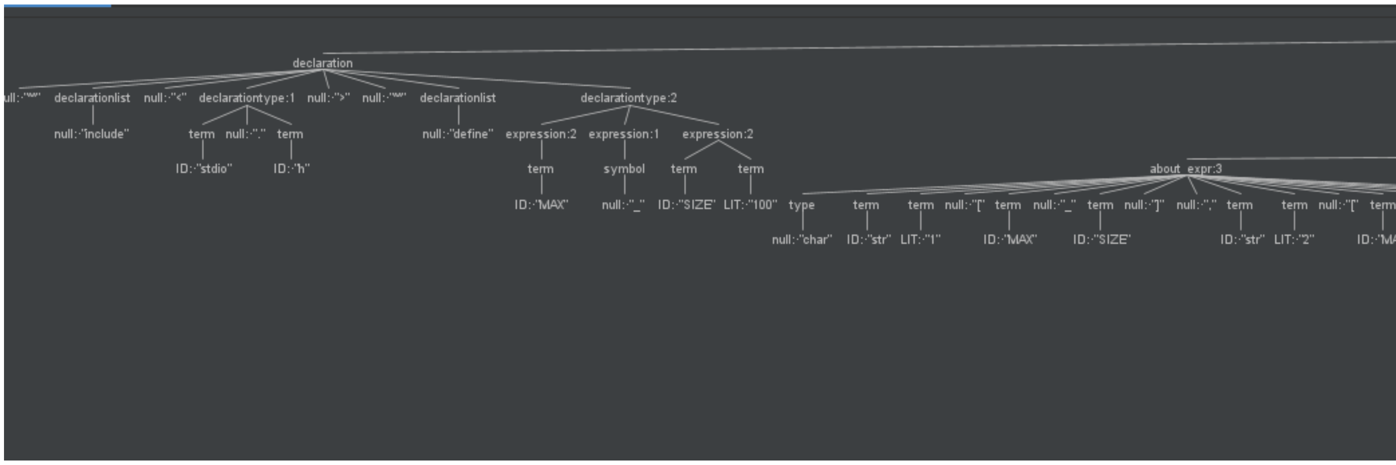


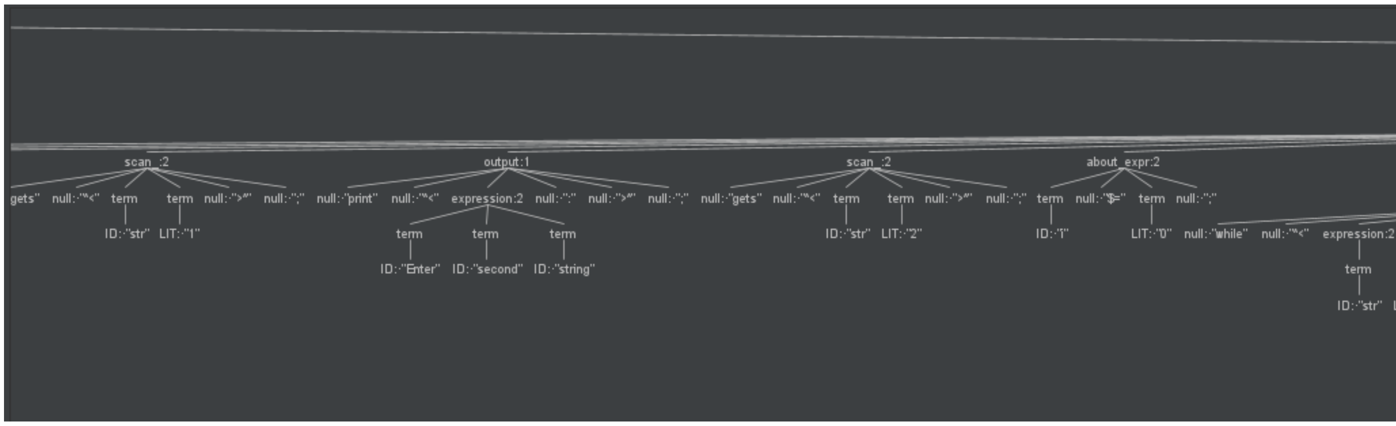
5.Whileloop and array

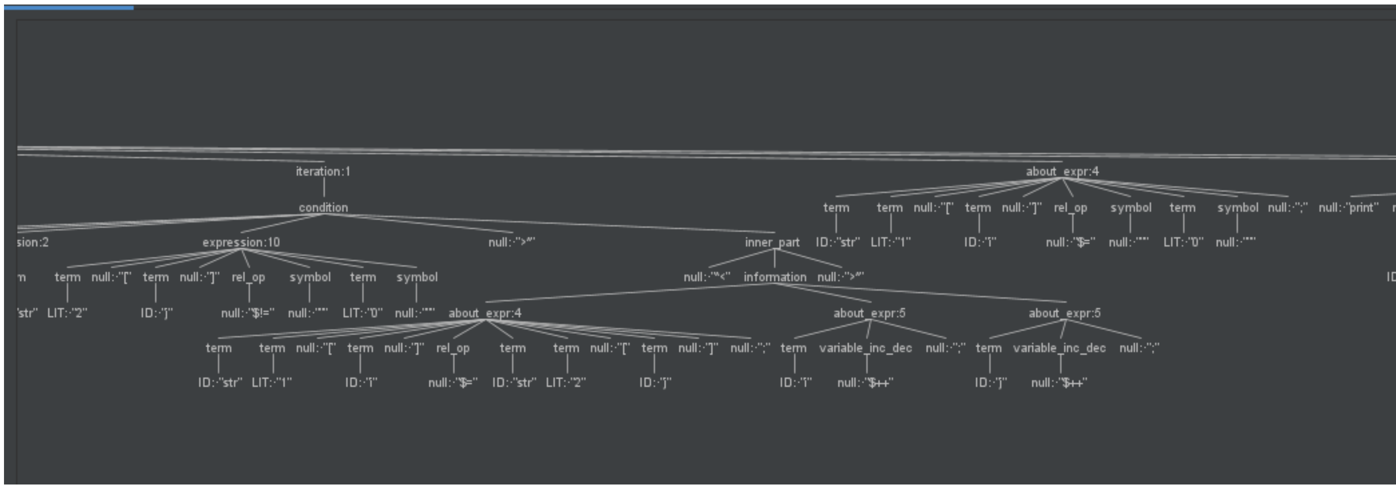
```
 ^^include <stdio.h>

^^define MAX_SIZE 100
integer main()
^<
char str1[MAX_SIZE], str2[MAX_SIZE];
integer i, j;
print^<Enter first string: >^;
gets^<str1>^;
print^<Enter second string: >^;
gets^<str2>^;
i$=0;
```

```
while^<str1[i] $!= "0">^
^<
i$++;
>^
j $= 0;
while^<str2[j] $!= "0">^
^<
str1[i] $= str2[j];
i$++;
j$++;
>^
str1[i] $= "0";
print^<Concatenated string $= $%integer": str1>^;
return 0;
>^
```

Tree:


declaration
ull::"*" declarationlist null::"<" declarationtype:1 null::">" null::"*" declarationlist declarationtype:2
null::"include" term null::"." term null::"define" expression:2 expression:1 expression:2
ID::"stdio" ID::"h" term symbol term term about_expr:3
ID::"MAX" null::"_" ID::"SIZE" LIT::"100" type term term null::"[" term null::"_" term null::"]" null::"," term term null::"[" term
null::"char" ID::"str" LIT::"1" ID::"MAX" ID::"SIZE" ID::"str" LIT::"2" ID::"MA


scan_:2 output:1 scan_:2 about_expr:2
gets" null::"*<" term term null::">*" null::";" null::"print" null::"*<" expression:2 null::":" null::">*" null::";" null::"gets" null::"*<" term term null::">*" null::";" term null::"$=" term null::";"
ID::"str" LIT::"1" term term term ID::"str" LIT::"2" ID::"i" LIT::"0" null::"while" null::"*<" expression:2
ID::"Enter" ID::"second" ID::"string" term
ID::"str"

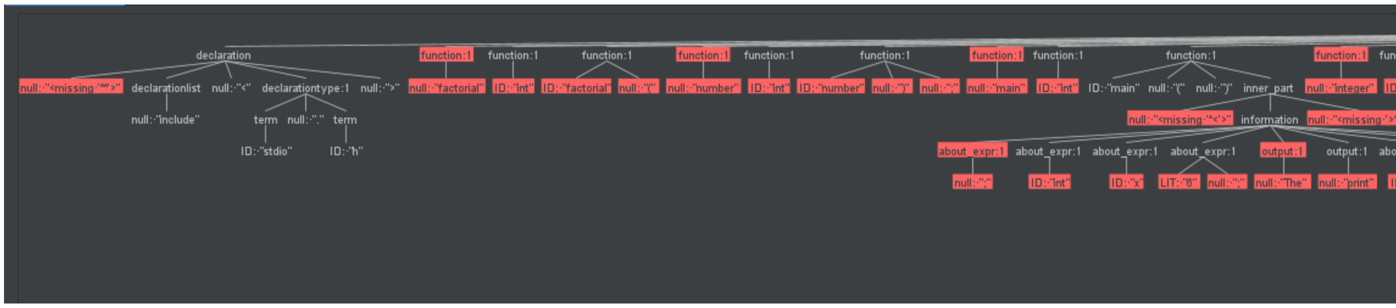


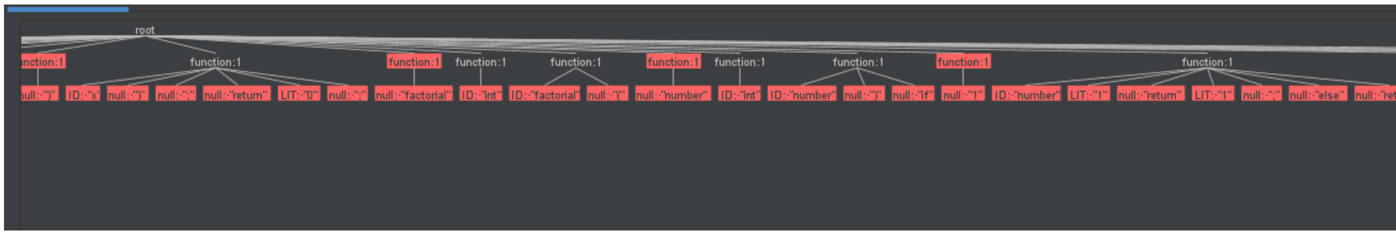
Incorrect sample input

1.Function-recursion

```
 ##include <stdio.h>

int factorial(int number);
int main()
{
int x = 6;
print {The factorial of %integer is %integer: x factorial(x)};
return 0;
}
int factorial(int number)
{
if { number == 1}
{
return 1;
}
else
{
return number * factorial(number - 1);
}
}
```

Tree:

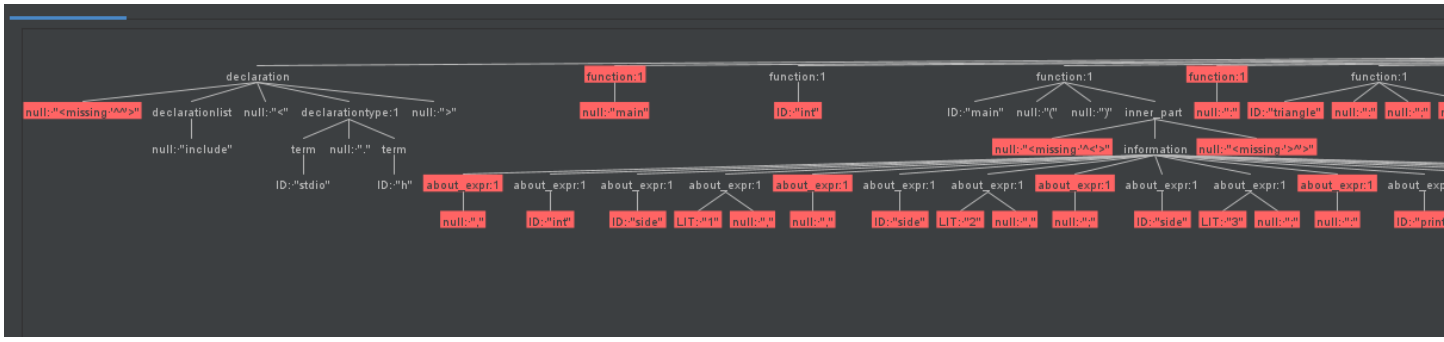


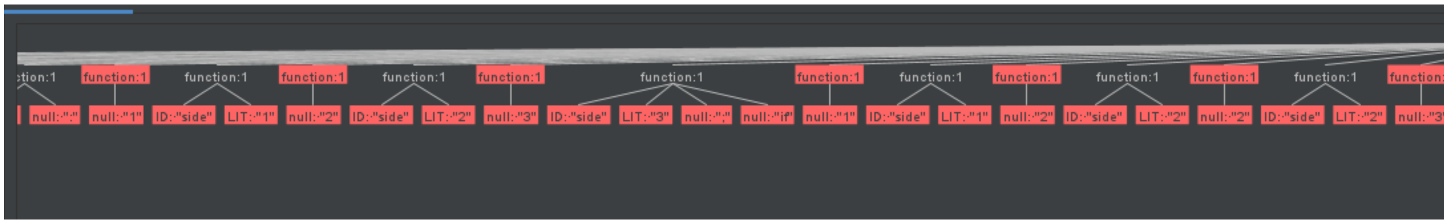


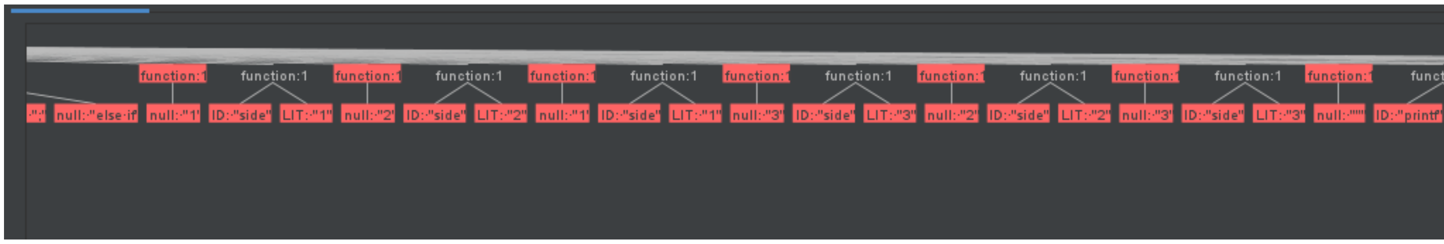
2.if else

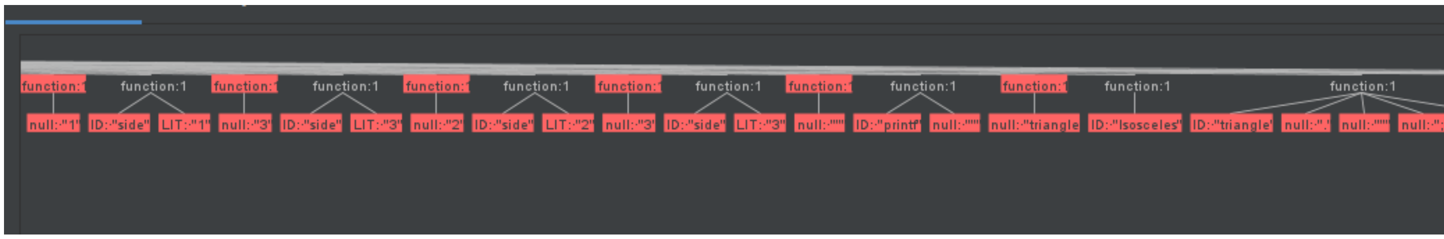
```
##include <stdio.h>

int main()
{
int side1, side2, side3;
printf{Enter three sides of triangle: };
scan{%int$%int$%int: side1 side2 side3};
if{side1 == side2 && side2 == side3}
{
printf{"Equilateral triangle."};
}
else if{side1==side2 || side1==side3 || side2==side3}
{
printf{"Isosceles triangle."};
}
else
{
printf{"Scalene triangle."};
}
return 0;
}
```

Tree:





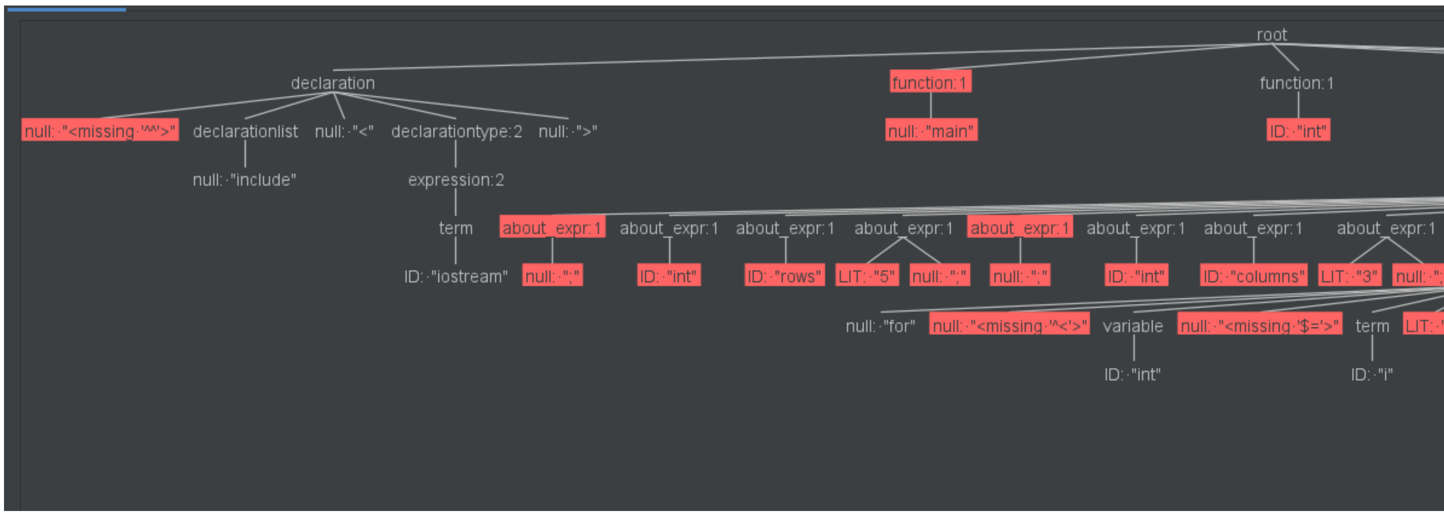


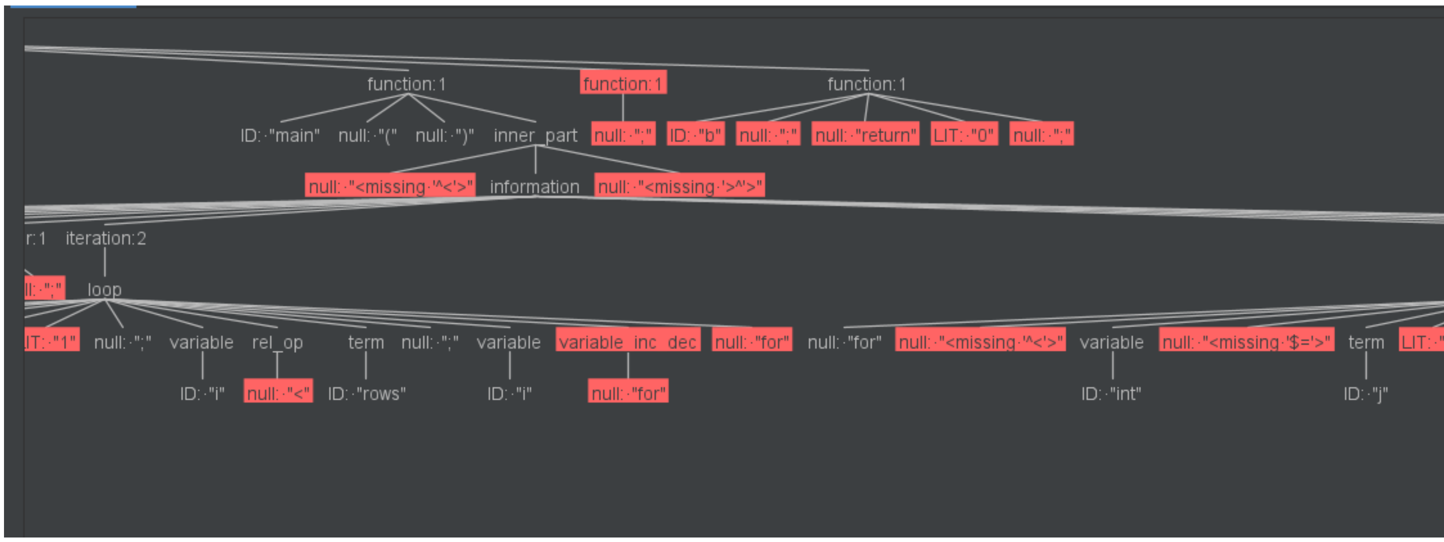


3. Nested-loop

```
 ##include <iostream>

int main ()
{
int rows = 5;
int columns = 3;
for {int i = 1; i <= rows; ++i }
{
for {int j = 1; j <= columns; ++j}
{
printf{a b};
}
}
return 0;
}
```

Tree:

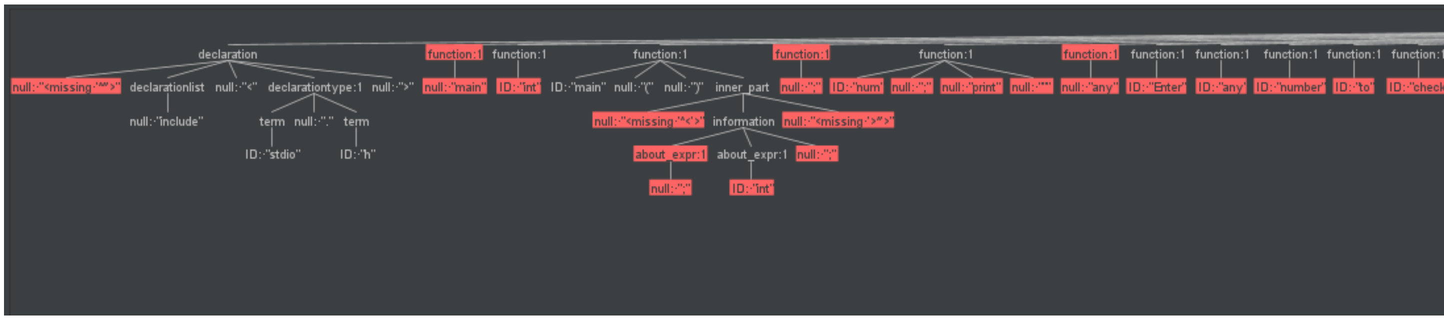


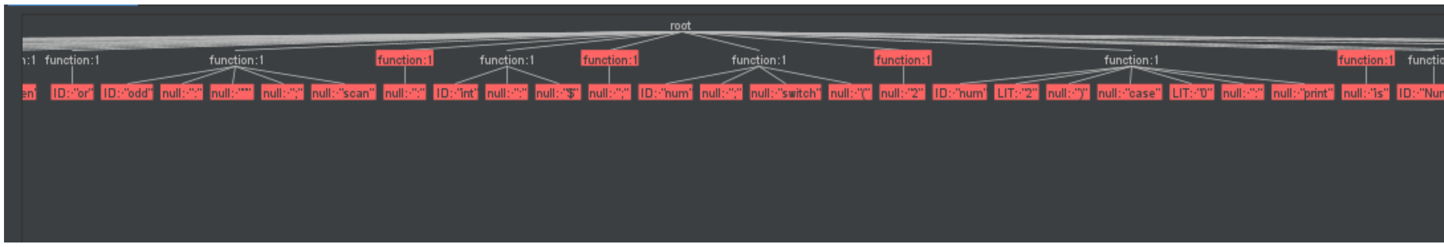


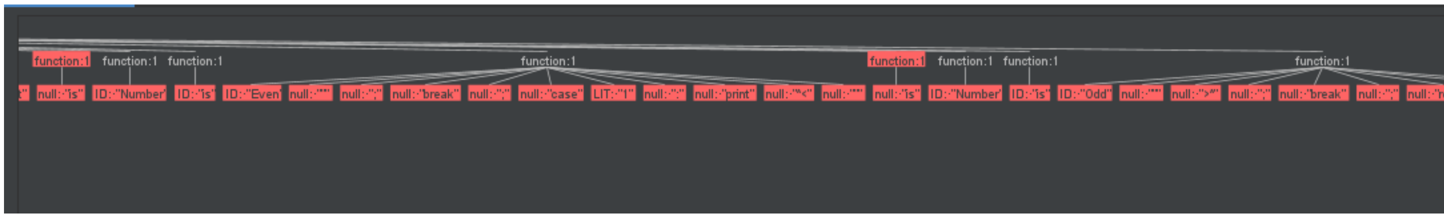
4. Switch case

```
 ##include <stdio.h>

int main()
{
int num;
print{"Enter any number to check even or odd: "};
scan{%int: $num};
switch(num % 2)
{
case 0:
{
print{Number is Even"};
break;
}
case 1:
{
print^<"Number is Odd">^;
break;
}
}
return 0;
}
```

Tree:



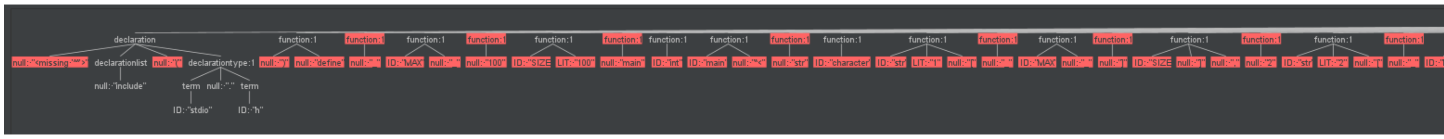


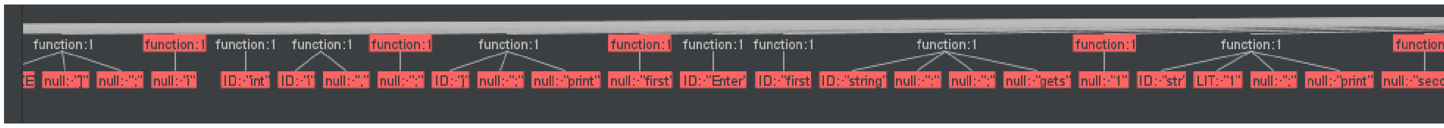


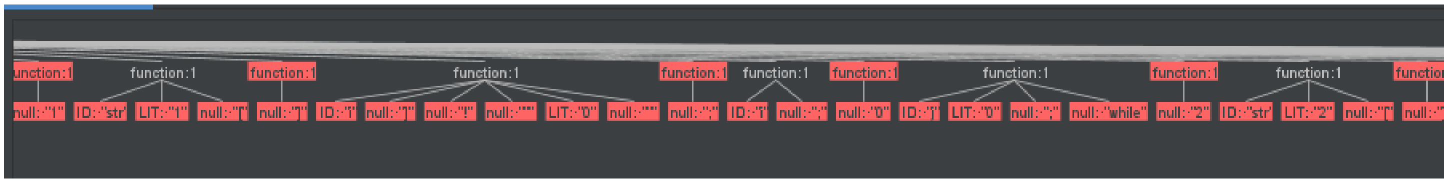
5.While loop and array

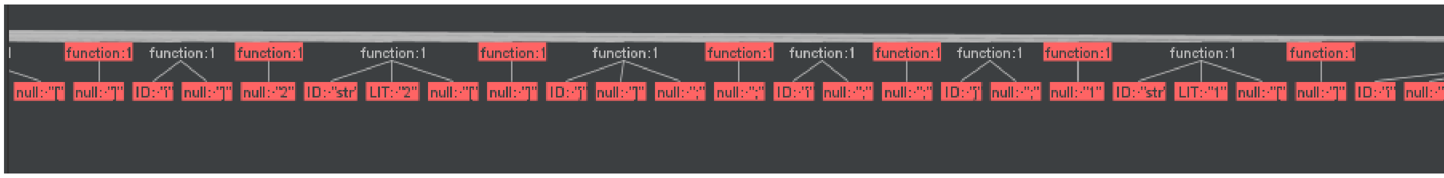
```
 ##include (stdio.h)
##define MAX_SIZE 100

int main{}
^<
character str1[MAX_SIZE], str2[MAX_SIZE];
int i, j;
print{Enter first string: };
gets{str1};
print{Enter second string: };
gets{str2};
i=0;
while{str1[i] != "0"}
{
i++;
}
j = 0;
while{str2[j] != "0"}
{
str1[i] = str2[j];
i++;
j++;
}
str1[i] = "0";
print{Concatenated string = %int": str1};
return 0;
}
```

Tree:




function:1
null::"]"
null::";"
null::"i"
ID::"int"
ID::"i"
null::","
null::";"
ID::"j"
null::";"
null::"print"
null::"first"
ID::"Enter"
ID::"first
ID::"string"
null::":"
null::";"
null::"gets"
null::"1"
ID::"str
LIT::"1"
null::";"
null::"print"


function:1
null::"1"
ID::"str
LIT::"1"
null::"["
null::"]"
ID::"i"
null::"]"
null::"!"
null::""
LIT::"0"
null::""
null::";"
ID::"i"
null::";"
null::"0"
ID::"j"
LIT::"0"
null::";"
null::"while"
null::"2"
ID::"str
LIT::"2"
null::"["


function:1
null::"["
null::"]"
ID::"i"
null::"]"
null::"2"
ID::"str
LIT::"2"
null::"["
null::"]"
ID::"j"
null::"]"
null::";"
null::";"
ID::"i"
null::";"
null::";"
ID::"j"
null::";"
null::"1"
ID::"str
LIT::"1"
null::"["
null::"]"
ID::"i"