East West University

Project Report

Course Code & Title: CSE375 Compiler Design Project

Section: 01

Semester: FALL'21

Submitted to: -

Dr. Shamim H Ripon

Professor, Department of CSE

East West University

Submitted by:

Name	ID
Hasibul Islam	2018-1-60-122
Ashiqur Rahman Anik	2018-1-60-149
Md. Jannatul Huq	2018-1-60-224

Date of Submission: 14-01-22

Project Title:

Generating a Parser for a customized (imaginary) programming language using ANTLR.

Project Description:

The grammar is designed to support the language type which includes:

1. The grammar starts from root and a declaration function with include and define feature is set up here.

2. A main function including types of expressions, conditional statements, iteration statements, logical conditions, integers, reading and writing variables, break statement are also set up here.

3. The declaration consists at least (declare_include | declare_define)+ or more;

4. The declare_include function consists of '[' 'include' '(' declarationtype ')' ']';

5. The declare_define function consists of '[' 'define' LIT 'as' ID ']';

6. The declarationtype function consists of ID '.' ID ;

7. The statement function consists of :(expressionstmt | selectionstmt | iterationstmt | statement_return | outputstmt | inputstmt | breakstmt)+;

8. The expressionstmt cotains the following expr ',' typeSpecifier ';' ;

9. The expr contains the following expr binop expr | expr relop expr | expr logical_op expr | '(' expr ')' | term ;

10. The statement_return contains the following pattern 'return' expr ';' | 'return' term ';' ;

11. The binop contains the following pattern '+' | '-' | '*' | '/' ;

12. The relop function follows the pattern : '==' | '!=' | '<=' | '<' | '>' | '>=' | '=' ;

13. The logical_op contains the following pattern: 'and' | 'or' | 'not' ;

14. The selectionstmt statement contains the pattern: 'if' '[' expr ']' block | 'if' '[' expr ']' block ('elif' '[' expr ']' block)* 'else' block ;

15.The breakstmt contains the following pattern of : 'break' ;

16.The iterationstmt statement follows the pattern: whilestmt | loopstmt ;

17. The whilestmt contains the following pattern: 'while' '[' expr ']' block ;

18. The loopstmt follows the pattern : 'loop"['loopexpr']' block ;

19. The loopexpr follows the pattern: var '='term 'to' var '=' term ',' 'increment' 'by' term ;

20. The outputstmt contains the pattern : 'write' ':' expr ';' ;

21. The inputstmt contains the following pattern : 'read' ':' var ';' ;

The syntax of the grammar, the input and its corresponding parse tree is given below using upper given description.

Grammar:

```
grammar prog;
root: declaration function;
declaration: (declare_include | declare_define)+ ;
declare_include: '[' 'include' '(' declarationtype ')' ']';
declare_define: '[' 'define' LIT 'as' ID ']';
declarationtype: ID '.' ID ;
function: 'main' '[' ']' ':' block;
block: '{' statement '}' ;
statement:(
 expressionstmt
| selectionstmt
| iterationstmt
| statement_return
| outputstmt
| inputstmt
| breakstmt
)+
;
expressionstmt : expr ',' typeSpecifier ';' ;
```

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

statement_return : 'return' expr ';' | 'return' term ';' ;

binop : '+' | '-' | '*' | '/' ;

relop : '==' | '!=' | '<=' | '<' | '>' | '>=' | '=' ;

logical_op : 'and' | 'or' | 'not' ;

selectionstmt : 'if' '[' expr ']' block | 'if' '[' expr ']' block ('elif' '[' expr ']' block)* 'else' block ;

breakstmt : 'break' ;

iterationstmt : whilestmt | loopstmt ;

whilestmt : 'while' '[' expr ']' block ;

loopstmt : 'loop"['loopexpr']' block ;

loopexpr : var '='term 'to' var '=' term ',' 'increment' 'by' term ;

outputstmt : 'write' ':' expr ';' ;

inputstmt : 'read' ':' var ';' ;

var : ID ;

incr_op : '++'| '--' ;

term : ID | LIT ;

typeSpecifier : 'integer' | 'character' | 'float' ;

ID : [a-zA-Z]+ ;

LIT : [0-9]+ ;

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

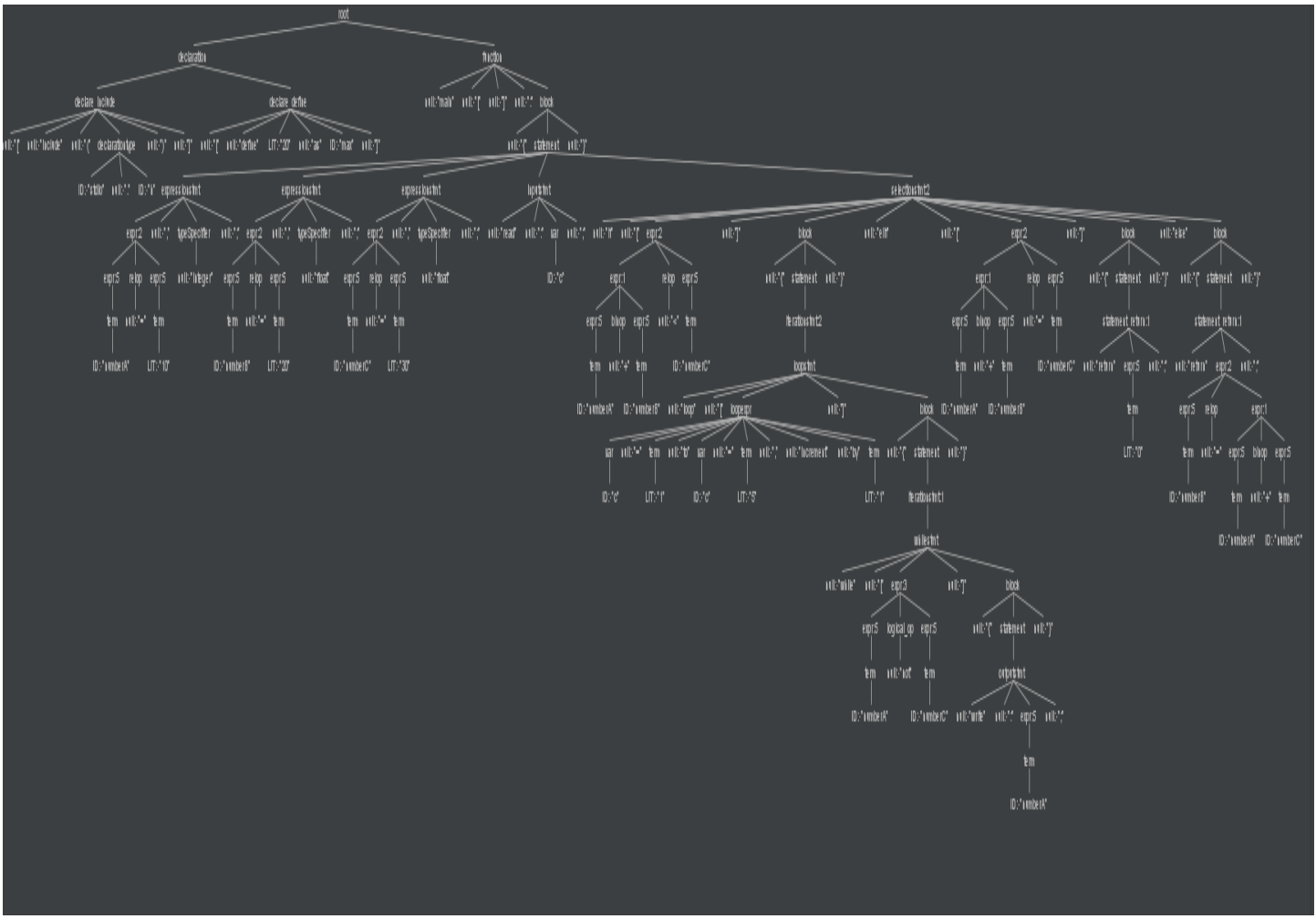
Correct input;

[include (stdio.h)]

[define 20 as max]

```
main[]:
{
 numberA  = 10, integer;
 numberB  = 20, float;
 numberC=30, float;
 read: c;
if[numberA  + numberB  < numberC]
{loop[c=1 to c = 5,increment by 1]
{while[numberA  not numberC]
{write: numberA ;
}
}
}
elif[numberA  + numberB  = numberC ]
{ return 0;
}
else{
        return numberB =numberA  + numberC;
}
}
```

Parse tree for correct input:



Parse tree for wrong input:

```
#include (stdio.h)]
[define 20 as max]
main[]:
{numberA  = 10, integer;
 numberB  = 20, float;
 numberC=30, float;
 read: c;
if[numberA  + numberB  < numberC]
{loop[integer c=1 to c = 5,increment by 1]
{while[numberA  not numberC]
{write: numberA ;
}
}
}
elif[numberA  + numberB  = numberC ]
 {
return 0;
}
else
{return *numberB =numberA  + numberC;
}
}
```

Wrong parse tree: