CAUSWE Compiler, 2021

Compiler Project #2 Documentation

Class#01 Team#59 안재형**(20171248)**

Class#02 Team#56 구건모**(20181856)**

How to run

Requirements

- Python >= 3.8.2
- Ubuntu or MacOS or Windows with Python

Execution

run.sh file_name

run.sh

```
#!/bin/bash

python3 analyzer/lexical_analyzer.py $1 $1.lex_out
python3 analyzer/syntax_analyzer.py $1.lex_out
```

“analyzer/lexical_analyzer.py” (which is made from Assignment #1) reads “file_name” and saves it’s output to “file_name.lex_out”.

Then, “analyzer/syntax_analyzer.py” reads “file_name.lex_out” and checks if it’s syntactically acceptable or not.

Context Free Grammar G

What is Changed?

1. Start Symbol
    - There are three production rules from the starting symbol CODE. So we binded them with CODE0.

2. Removed Ambiguity of EXPR
    - An ambiguity exists between addsub and multdiv.
      Ex) “id addsub id multdiv id” can have 2 parse trees.
    - So we set a disambiguiting rule that multdiv has a higher priority than addsub.

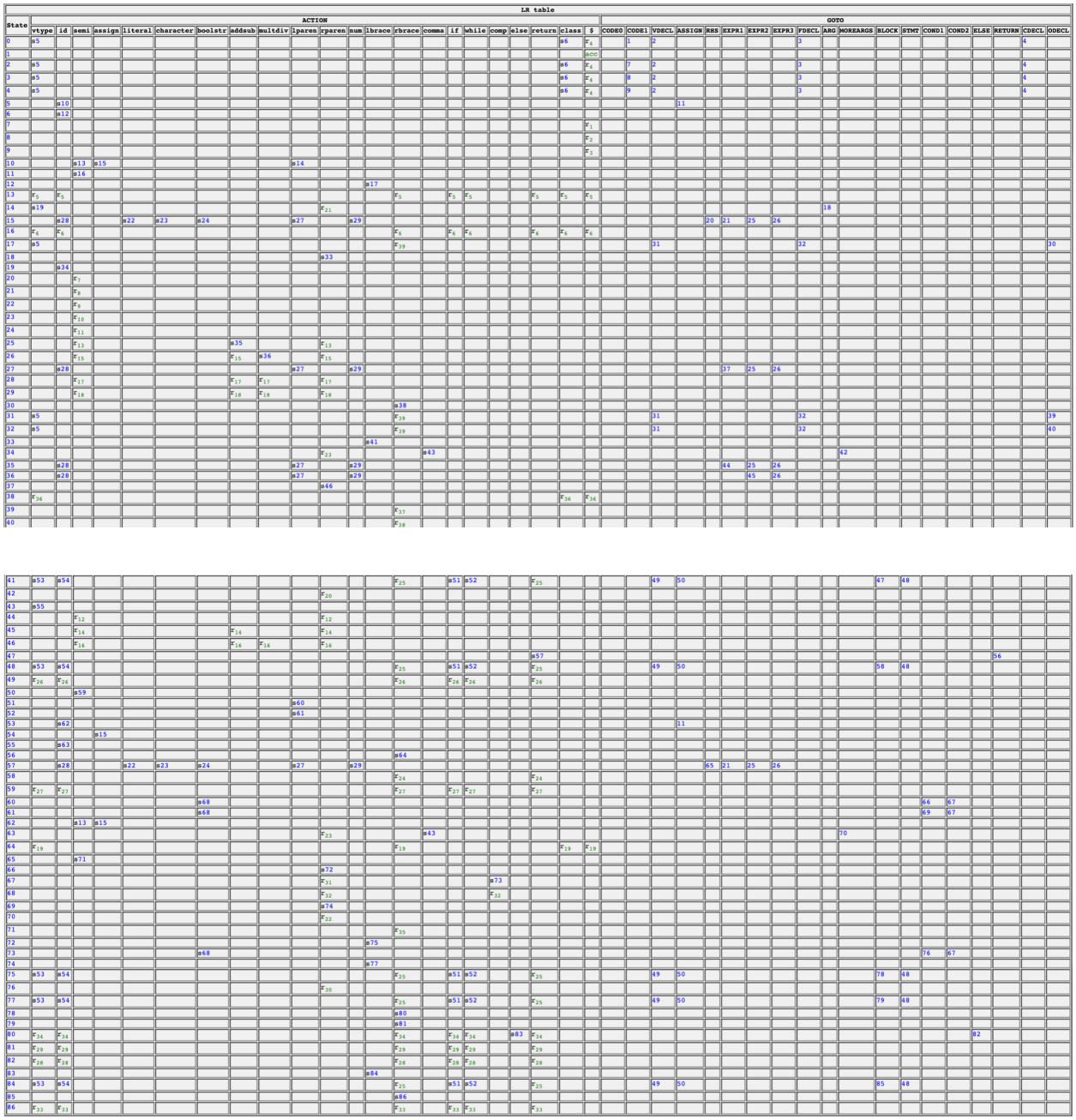
3. Removed Ambiguity of EXPR and COND
    - At EXPR, ambiguity rules exist between addsub and multdiv.
      Ex) “id addsub id addsub id”
    - So we set a disambiguiting rule that right addsub and multdiv has a higher priority than left addsub and multdiv.

    - At COND, ambiguity rule exists between comps.
      Ex) “boolstr comp boolstr comp boolstr”
    - So we set a disambiguiting rule that right comp has a higher priority than left comp.

CFG G (Ambiguity Removed)

CODE0 -> CODE1
CODE1 -> VDECL CODE1
CODE1 -> FDECL CODE1
CODE1 -> CDECL CODE1
CODE1 -> ''
VDECL -> vtype id semi
VDECL -> vtype ASSIGN semi
ASSIGN -> id assign RHS
RHS -> EXPR1
RHS -> literal
RHS -> character
RHS -> boolstr
EXPR1 -> EXPR2 addsub EXPR1
EXPR1 -> EXPR2
EXPR2 -> EXPR3 multdiv EXPR2
EXPR2 -> EXPR3
EXPR3 -> lparen EXPR1 rparen
EXPR3 -> id
EXPR3 -> num
FDECL -> vtype id lparen ARG rparen lbrace BLOCK RETURN rbrace
ARG -> vtype id MOREARGS
ARG -> ''
MOREARGS -> comma vtype id MOREARGS
MOREARGS -> ''
BLOCK -> STMT BLOCK
BLOCK -> ''
STMT -> VDECL
STMT -> ASSIGN semi
STMT -> if lparen COND1 rparen lbrace BLOCK rbrace ELSE
STMT -> while lparen COND1 rparen lbrace BLOCK rbrace
COND1 -> COND2 comp COND1
COND1 -> COND2
COND2 -> boolstr
ELSE -> else lbrace BLOCK rbrace
ELSE -> ''
RETURN -> return RHS semi
CDECL -> class id lbrace ODECL rbrace
ODECL -> VDECL ODECL
ODECL -> FDECL ODECL
ODECL -> ''

SLR Parsing Table from CFG G



LR table

State	ACTION																				GOTO																			
	vtype	id	semi	assign	literal	character	boolstr	addsub	multdiv	lparen	rparen	num	lbrace	rbrace	comma	if	while	comp	else	return	class	$	CODE0	CODE1	VDECL	ASSIGN	RHS	EXPR1	EXPR2	EXPR3	FDECL	ARG	MOREARGS	BLOCK	STMT	COND1	COND2	ELSE	RETURN	CDECL	ODECL
0	s5																			s6	F_4			1	2						3								4	
1																					acc																				
2	s5																			s6	F_4			7	2						3								4	
3	s5																			s6	F_4			8	2						3								4	
4	s5																			s6	F_4			9	2						3								4	
5		s10																			F_1					11															
6		s12																																							
7																					F_1																			
8																					F_2																			
9																					F_3																			
10			s13	s15						s14																															
11			s16																																						
12													s17																												
13	F_5	F_5												F_5		F_5	F_5			F_5	F_5	F_5																			
14	s19										F_{21}																					18									
15		s28			s22	s23	s24			s27		s29															20	21	25	26												
16	F_6	F_6												F_6		F_6	F_6			F_6	F_6	F_6																			
17	s5													F_{39}											31						32									30
18											s33																														
19		s34																																						
20			F_7																																					
21			F_8																																					
22			F_9																																					
23			F_{10}																																					
24			F_{11}																																					
25			F_{13}					s35			F_{13}																													
26			F_{15}					F_{15}	s36		F_{15}																														
27		s28								s27		s29															17	25	26												
28			F_{17}					F_{17}	F_{17}		F_{17}																														
29			F_{18}					F_{18}	F_{18}		F_{18}																														
30														s38																											
31	s5													F_{39}											31						32									39
32	s5													F_{39}											31						32									40
33													s41																											
34											F_{23}				s43																		42							
35		s28								s27		s29															44	25	26												
36		s28								s27		s29																45	26												
37											s46																														
38	F_{16}																			F_{16}	F_{16}																			
39														F_{37}																											
40														F_{38}																											
41	s53	s54												F_{25}		s51	s52			F_{25}					49	50								47	48						
42											F_{20}																														
43	s55																																							
44			F_{12}								F_{12}																														
45			F_{14}					F_{14}			F_{14}																														
46			F_{16}					F_{16}	F_{16}		F_{16}																														
47																				s_{57}																		F_{56}		
48	s53	s54												F_{25}		s51	s52			F_{25}					49	50								58	48						
49	F_{26}	F_{26}												F_{26}		F_{26}	F_{26}			F_{26}																					
50			s59																																					
51										s60																														
52										s61																														
53		s62																							11															
54				s15																																				
55		s63																																					
56														s64																										
57		s28			s22	s23	s24			s27		s29															65	21	25	26												
58														F_{24}						F_{24}																					
59	F_{27}	F_{27}												F_{27}		F_{27}	F_{27}			F_{27}																					
60							s68																																		
61							s68																																		
62			s13	s15																																				
63											F_{23}				s43																		70							
64	F_{19}													F_{19}							F_{19}	F_{19}																		
65			s71																																					
66											s72																													
67											F_{31}							s73																						
68											F_{32}							F_{32}																						
69											s74																													
70											F_{22}																														
71														F_{35}																											
72													s75																											
73							s68																															F_{76}	F_{67}		
74													s77																											
75	s53	s54												F_{25}		s51	s52			F_{25}					49	50								78	48						
76											F_{30}																														
77	s53	s54												F_{25}		s51	s52			F_{25}					49	50								79	48						
78														s80																											
79														s81																											
80	F_{34}	F_{34}												F_{34}		F_{34}	F_{34}		s83	F_{34}																		F_{82}			
81	F_{29}	F_{29}												F_{29}		F_{29}	F_{29}			F_{29}																					
82	F_{28}	F_{28}												F_{28}		F_{28}	F_{28}			F_{28}																					
83													s84																											
84	s53	s54												F_{25}		s51	s52			F_{25}					49	50								85	48						
85														s86																											
86	F_{33}	F_{33}												F_{33}		F_{33}	F_{33}			F_{33}																					

+ For Addition

We made a javascript script that parses http://jsmachines.sourceforge.net/machines/slr.html 's LR Table and converts it into python dict.

```
const table = document.getElementById("lrTableView")
const thead = Array.from(table.getElementsByTagName("thead")[0].children[2].children)
const rows = Array.from(table.getElementsByTagName("tbody")[0].children)
var dictionary = rows.map((rowHTML)=>{
   var result = {}
   const row = Array.from(rowHTML.children)
   row.forEach((colHTML, colIndex)=>{
      if(colIndex===0){
         return
      } else if (colHTML.innerHTML == " ") {
         result[thead[colIndex-1].innerText] = null
      } else {
         result[thead[colIndex-1].innerText] = colHTML.innerText
      }
   })
   return result
})
dictionary = JSON.stringify(dictionary)
dictionary = dictionary.replaceAll("null", "None")
console.log(dictionary)
```

This script runs on Chrome/Developer Tools/Sources/Snippets

How this Syntax Analyzer Work

Architecture

There’s a Stack class. This class is a normal stack that has a push and pop method. It will store state and left substring.

There’s a SLRtable class. This class contains a SLR-parsing table to parse given CFG. If it gets a symbol, The corresponding symbol of the state that the first number of stacks points to returns Decision. And follow Decision like shift and goto or reduce. Of course, if the table's return is None which means it can’t choose any decision, it prints a fail and reports an error.

There's a CFG class. This class contains list when reduced by following rules. List return grammars to change and how much pop the stack.

For the main function,

1. It only reads Token in its output from the lexical analyzer.
2. Then push the sub string and state in each stack by moving pipe.
3. Repeat it until the table returns ‘acc’ or ‘None’.
4. If it is 'acc', it prints 'ACCEPT', if it is 'None', it prints 'Fail', and it prints an error report.

Test Inputs/Outputs

Successfully Accepted Tokens

input file(test.txt)	.lex_out file (test.txt.lex_out)	console

int a;
int c() {
int a;
return a;
}
	vtype int
	id a
	semi ;
	vtype int
	id c
	lparen (
	rparen)
	lbrace {
	vtype int
	id a
	semi ;
	return return
	id a
	semi ;
	rbrace }
		Lexical Analyzer : Accepted
		Lexical Analyzer : Token file saved at test.txt.lex_out successfully
		Syntax Analyzer : Accepted

input file(test.txt)	.lex_out file (test.txt.lex_out)	console

int divide(int a, int b) {
  int result;
  if(true == true) {
  result = 0;
  } else {
  result = a/b;
  }
  return c;
}
	vtype int
	id divide
	lparen (
	vtype int
	id a
	comma ,
	vtype int
	id b
	rparen)
	lbrace {
	vtype int
	id result
	semi ;
	if if
	lparen (
	boolstr true
	comp ==
	boolstr true
	rparen)
	lbrace {
	id result
	assign =
	num 0
	semi ;
	rbrace }
	else else
	lbrace {
		Lexical Analyzer : Accepted
		Lexical Analyzer : Token file saved at test.txt.lex_out successfully
		Syntax Analyzer : Accepted

	id result assign = id a multdiv / id b semi ; rbrace } return return id c semi ; rbrace }	

input file(test.txt)	.lex_out file (test.txt.lex_out)	console

int divide(int a, int b) {
  int result;
  return result;
}
 	
class Main {
  int state = 0;
 	
  int main(int arg1, int arg2) {
  int state = 1;
  int result = 0;
  int a = 20;
  int b = 2;
  char c = 't';
  char str = "some_string";
 	
  if(true == false) {
  result = a/(a+b/a);
  result = a/b;
  } else {
  result = -1;
  }
 	
  return result;
  }
}
	vtype int
	id divide
	lparen (
	vtype int
	id a
	comma ,
	vtype int
	id b
	rparen)
	lbrace {
	vtype int
	id result
	semi ;
	return return
	id result
	semi ;
	rbrace }
	class class
	id Main
	lbrace {
	vtype int
	id state
	assign =
	num 0
	semi ;
	vtype int
	id main
	lparen (
	vtype int
	id arg1
	comma ,
	vtype int
	id arg2
	rparen)
	lbrace {
	vtype int
	id state
	assign =
	num 1
	semi ;
	vtype int
		Lexical Analyzer : Accepted
		Lexical Analyzer : Token file saved at test.txt.lex_out successfully
		Syntax Analyzer : Accepted

	id result assign = num 0 semi ; vtype int id a assign = num 20 semi ; vtype int id b assign = num 2 semi ; vtype char id c assign = character 't' semi ; vtype char id str assign = literal "some_string" semi ; if if lparen (boolstr true comp == boolstr false rparen) lbrace { id result assign = id a multdiv / lparen (id a addsub + id b multdiv / id a rparen) semi ; id result assign = id a multdiv / id b semi ; rbrace } else else lbrace { id result assign = num -1 semi ; rbrace } return return id result semi ;	

	rbrace } rbrace }	

Failed Tokens

input file(test.txt)	.lex_out file (test.txt.lex_out)	console

int divide(int a, int b) {
  int result;
  if(b == 0) {
  result = 0;
  } else {
  result = a/b;
  }
  return c;
}
	vtype int
	id divide
	lparen (
	vtype int
	id a
	comma ,
	vtype int
	id b
	rparen)
	lbrace {
	vtype int
	id result
	semi ;
	if if
	lparen (
	id b
	comp ==
	num 0
	rparen)
	lbrace {
	id result
	assign =
	num 0
	semi ;
	rbrace }
	else else
	lbrace {
	id result
	assign =
	id a
	multdiv /
	id b
	semi ;
	rbrace }
	return return
	id c
	semi ;
	rbrace }
		Lexical Analyzer : Accepted
		Lexical Analyzer : Token file saved at test.txt.lex_out successfully
		Syntax Analyzer : Failed
		Syntax Analyzer : 15's token lparen can't choice id