1. BNF (without considering left recursion)

```
<code> ::= program <identifier> program_begin <statements> program_end

<statements> ::= <statement> | <statement><statements> | ε
<statement> ::= <if_stmt> | <loop_stmt> | <assign_stmt> | <fcall_stmt>
<if_stmt> ::= if ( <relational_expr> ) <block> <elses_stmt>
<elses_stmt> ::= else <block> | elseif ( <relational_expr> ) <block> <elses_stmt> | ε
<loop_stmt> ::= <while_stmt> | <for_stmt>
<while_stmt> ::= while ( <relational_expr> ) <block>
<for_stmt> ::= for ( <for_init> ; <for_termination> ; <for_update> ) <block>
<for_init> ::= <assign_stmt> | ε
<for_termination> ::= <relational_expr> | ε
<for_update> ::= <unary_expr> | ε
<assign_stmt> ::= <type><identifier><assign_expr> ;
<fcall_stmt> ::= display(<literal>) ;
<block> ::= begin <statements> end

<expr> ::= <assign_expr> | <relational_expr> | <add_expr> | <mul_expr> | <unary_expr> |
<primary_expr>
<assign_expr> ::= <assign_operator><expr><declator> | ε
<relational_expr> ::= <expr><relational_operator><expr>
<add_expr> ::= <expr><add_operator><expr>
<mul_expr> ::= <expr><mul_operator><expr>
<unary_expr> ::= <identifier><unary_operator>
<primary_expr> ::= (<expr>) | <identifier> | <number> | <literal> | ε
<declator> ::= , <identifier><assign_expr> | ε

<assign_operator> ::= =
<relational_operator> ::= < | > | <= | >= | ==
<add_operator> ::= +
<mul_operator> ::= *
<unary_operator> ::= ++

<type> ::= integer | ε
<literal> ::= "<letters>"
<number> ::= <integer> | ε
<identifier> ::= <letter><id>

<letters> ::= <letter> | <letter><letters>
<letter> ::= "A" | "B" | ... | "y" | "z" | "_" | ε
<integer> ::= <digits>
<digits> ::= <digit> | <digit><digits>
<digit> ::= 0 | 1 | ... | 8 | 9
<id> ::= _<id><terminal_id> | <letter><id><terminal_id> | <digit><id><terminal_id> | ε
<terminal_id> ::= _ | <letter> | <digit>
```

2. FirstSet

| | |
|---|---|
| 'code | {program} |
| code | {program} |
| statements | {if,while,for,display,break,integer,'',identifier,;} |
| statement | {if,while,for,display,break,integer,'',identifier,;} |
| if_stmt | {if} |
| while_stmt | {while} |
| for_stmt | {for} |
| id_init | {integer,'',identifier} |
| assign_stmt | {integer,'',identifier,;} |
| fcall_stmt | {display} |
| break_stmt | {break} |
| block | {begin} |
| unary_expr | {identifier} |
| assign_expr | {=,,,''} |
| relational_expr | {identifier,number} |
| expr | {identifier,number} |
| add_expr | {identifier,number} |
| mul_expr | {identifier,number} |
| declator | {,,''} |
| relational_operator | {<,>,>=,<=,==} |
| type | {integer,''} |

3. FollowSet

| | |
|---|---|
| 'code | {$} |
| code | {$} |
| statements | {program_end, end, if, while, for, display, break, integer, identifier, ;} |
| statement | {program_end, end, if, while, for, display, break, integer, identifier, ;} |
| if_stmt | {program_end, end, if, while, for, display, break, integer, identifier, ;} |
| while_stmt | {program_end, end, if, while, for, display, break, integer, identifier, ;} |
| for_stmt | {program_end, end, if, while, for, display, break, integer, identifier, ;} |
| id_init | {;} |
| assign_stmt | {program_end, end, if, while, for, display, break, integer, identifier, ;} |
| fcall_stmt | {program_end, end, if, while, for, display, break, integer, identifier, ;} |
| break_stmt | {program_end, end, if, while, for, display, break, integer, identifier, ;} |
| block | {if, while, for, display, break, integer, identifier, ; , elseif, else, program_end, end} |
| unary_expr | {)} |
| assign_expr | {;} |
| relational_expr | {;, )} |
| expr | {+, *, ;, ), ','} |
| add_expr | {identifier, number} |
| mul_expr | {identifier, number} |
| declator | {;} |
| relational_operator | {identifier, number} |
| type | {identifier} |

4. DFA of LR(1)

- LR(1) Items

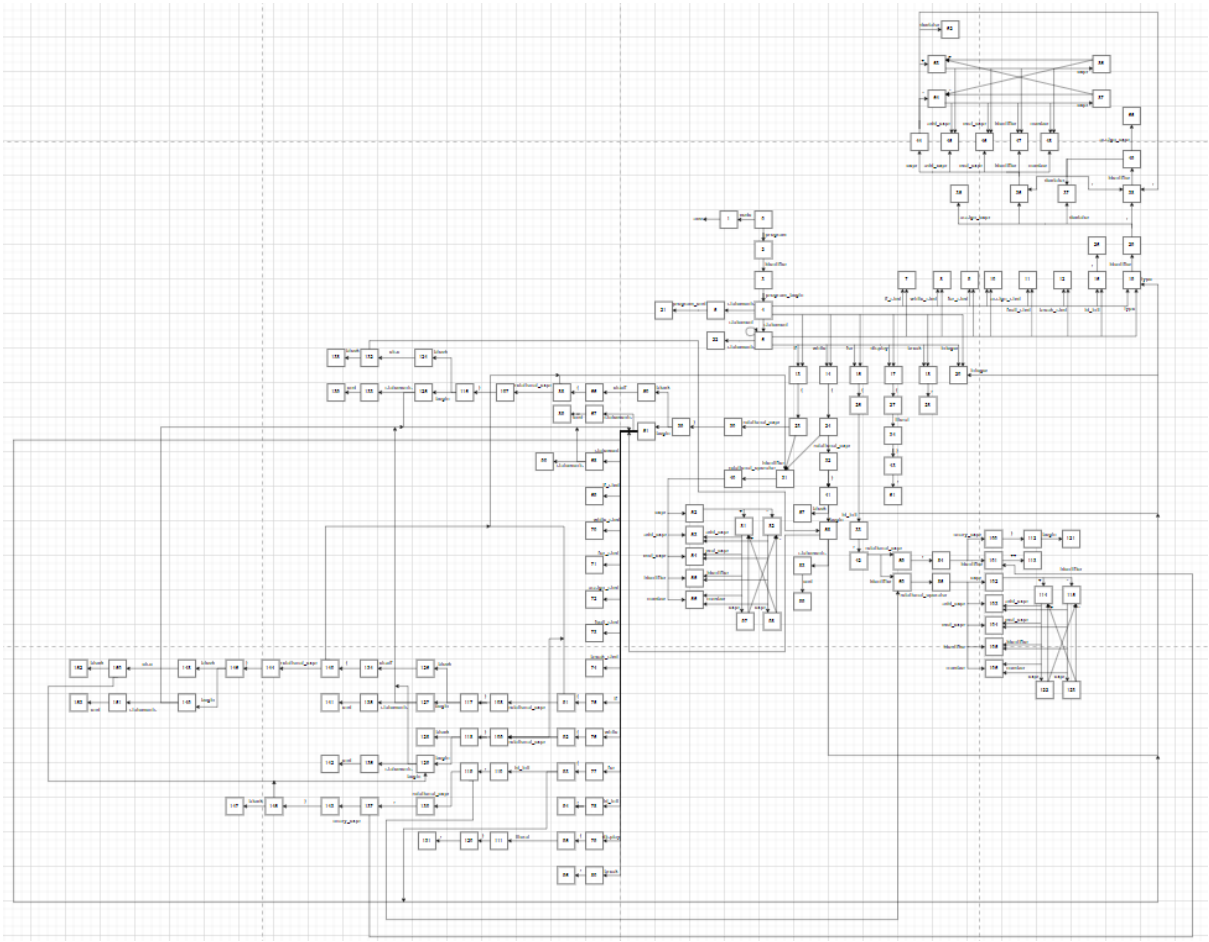
'code → code
code → program identifier program_begin statements program_end

statements → statement
statements → statement statements
statement → if_stmt
statement → while_stmt
statement → for_stmt
statement → assign_stmt
statement → fcall_stmt
statement → break_stmt
if_stmt → if ( relational_expr ) block
if_stmt → if ( relational_expr ) block elseif ( relational_expr ) block else block
if_stmt → if ( relational_expr ) block elseif ( relational_expr ) block
while_stmt → while ( relational_expr ) block
for_stmt → for ( id_init ; relational_expr ; unary_expr ) block
id_init → type identifier assign_expr
assign_stmt → id_init ;
fcall_stmt → display ( literal ) ;
break_stmt → break ;
block → begin statements end

unary_expr → identifier ++
assign_expr → = expr declator
assign_expr → declator
relational_expr → identifier relational_operator expr
expr → add_expr
expr → mul_expr
expr → identifier
expr → number
add_expr → expr + expr
mul_expr → expr * expr
declator → , identifier assign_expr
declator → ε

type → integer
type → ε

- DFA



5. Parsing Table

The Parsing Table is stored as ParsingTable.xlsx with DFA items and examples of parsing stacks of three test cases. Please check the file. However, the sheet name ‘ParsingStack - test (number)’ is not the same with the result of above DFA and source code. Those sheets are for testing purposes only to compare “relational_expr → identifier relational_operator expr” “relational_expr → expr relational_operator expr”. My parse took the first one, despite the remaining sheets following the second.