Tutorial
Syntax Analysis

Thang Huu Nguyen

September 2019

Question 1.

Given the description of a program in mC as follows:

A program in mC consists of many declarations, which are variable and function declarations.

A variable declaration starts with a type, which is int or float, then a comma-separated list of identifiers and ends with a semicolon.

A function declaration also start with a type and then an identifier, which is the function name, and then parameter declaration and ends with a body. The parameter declaration starts with a left round bracket '(' and a null-able semicolon-separated list of parameters and ends with a right round bracket ')'. Each parameter always starts with a type and then a comma-separated list of identifier. A body starts with a left curly bracket '{', follows by a null-able list of variable declarations or statements and ends with a right curly bracket '}'.

There are 3 kinds of statements: assignment, call and return. All statements must end with a semicolon. An assignment statement starts with an identifier, then an equal '=', then an expression. A call starts with an identifier and then follows by a null-able comma-separated list of expressions enclosed by round brackets. A return statement starts with a symbol 'return' and then an expression.

An expression is a construct which is made up of operators and operands. They calculate on their operands and return new value. There are four kinds of infix operators: '+', '-', '*' and '/' where '+' have lower precedence than '-' while '*' and '/' have the highest precedence among these operators. The '+' operator is right associative, '-' is non-associative while '*' and '/' is left-associative. To change the precedence, a sub-expression is enclosed in round brackets. The operands can be an integer literal, float literal, an identifier, a call or a sub-expression.

For example,

```
int a, b, c ;
float foo(int a ; float c, d) {
    int e ;
    e = a + 4;
    c = a * d / 2.0;
    return c + 1 ;
}
float goo (float a, b) {
    return foo(1, a, b) ;
}
```

The following tokens can be used for the grammar:

ID (for identifiers), **INTLIT** (for integer literals), **FLOATLIT** (for float literals), **INT**, **FLOAT**, **RETURN**, **LB** (for '{'), **RB** (for '}') , **SM** (for ';'), **CM** (for ','), **EQ** (for '='), **LP** (for '('), **RP** (for ')'), **ADD** (for '+'), **SUB** (for '-'), **MUL** (for '*'), **DIV** (for '/')

a. Write the grammar of a program in mC in BNF format.
Answer:
program → manydcls
manydcls → dcls manydcls | dcls
dcls → vardcls | funcdcls
vardcls → type idlist SM
type → INT | FLOAT
idlist → ID CM idlist | ID
funcdcls → type ID paradcls body
paradcls → LP paralist RP
paralist → para paratail |∈
paratail → SM para paratail |∈
para → type idlist
body → LB vardcl_stmt_list RB
vardcl_stmt_list → vardcl_stmt vardcl_stmt_list |∈
vardcl_stmt → vardcls | stmt
stmt → stmt_type SM
stmt_type → assign | call | return
assign → ID EQ exp
call → ID LP explist RP
explist → exp exptail |∈
exptail → CM exp exptail |∈
return → RETURN exp
exp → exp1 ADD exp | exp1
exp1 → exp1 SUB exp1 | exp2
exp2 → exp2 MUL exp3 | exp2 DIV exp3 | exp3
exp3 → INTLIT | FLOATLIT | ID | call | subexp
subexp → LP exp RP

b. Write a recognizer in ANTLR to detect if a mC program is written correctly or not
Answer: Answer to this question has been uploaded source code above.