解释器第三次实验报告

一、背景

- 姓名：余璞轩
- 学号：2014302580341
- 班级：卓越二班
- 提交日期：11月16日

任务：

- 设计并编制调试一个分析 CMM 程序结构的语法分析器.
- 加深对语法分析原理的理解和应用。

二、实验内容

1.编译方法说明

(1)文法

根据`《解释器构造》课程任务书`中给出的cmm语言的文法，可以根据g4文件的格式写出cmm在antlr工具中的语法文件。

```
grammar cmm;

program :
    (stmt)+
    ;

stmt :
    var_decl                #vardecl
    | if_stmt               #ifstmt
    | while_stmt            #whilestmt
    | break_stmt            #breakstmt
    | assign_stmt           #assignstmt
    | read_stmt             #readstmt
    | write_stmt            #writestmt
```

```
    | stmt_block            #stmtblock
    | LParen stmt RParen    #LParenStmtRParen
    ;

stmt_block :
    lcurly (stmt)* rcurly
    ;

var_decl :
    type var_list lineend
    ;

type :
    int_ |
    double_ |
    type lbracket intconstant rbracket
    ;

var_list :
    ident (comma ident)*
    ;

decl_assign :
    ident assign expr
    ;

if_stmt :
    if_ expr stmt |
    if_ expr stmt else_ stmt
    ;

while_stmt :
    while_ expr stmt
    ;

break_stmt :
    break_ lineend
    ;

read_stmt :
    read_ lparen ident rparen lineend |
    read_ lparen ident lbracket intconstant rbracket lineend
    ;

write_stmt :
    write_ lparen expr rparen lineend
    ;
```

```
assign_stmt :
    value assign expr lineend
    ;

value :
    ident |
    ident lbracket intconstant rbracket
    ;

constant :
    intconstant |
    realconstant |
    booleanconstant
    ;

expr :
    expr muldivmod expr |
    expr addmin expr |
    expr compop expr |
    lparen expr rparen |
    ident |
    constant |
    addmin expr
    ;

if_ :
    If # getIf
    ;
else_ :
    Else # getElse
    ;
while_ :
    While # getWhile
    ;
read_ :
    Read # getRead
    ;
write_ :
    Write # getWrite
    ;
int_ :
    Int # getInt
    ;
double_ :
    Double # getDouble
    ;
```

```
break_ :
    Break # getBreak
    ;

If : 'if' ;
Else : 'else' ;
While : 'while' ;
Read : 'read' ;
Write : 'write' ;
Int : 'int' ;
Double : 'double' ;
Break : 'break' ;

ident :
    Ident # getIdent
    ;
intconstant :
    IntConstant # getIntConstant
    ;
realconstant :
    RealConstant # getRealConstant
    ;
booleanconstant :
    BooleanConstant # getBooleanConstant
    ;
compop :
    CompOp # getCompop
    ;
muldivmod :
    MulDivMod # getMulDivMod
    ;
addmin :
    AddMin # getAddMin
    ;
lparen :
    LParen # getLParen
    ;
rparen :
    RParen # getRParen
    ;
lcurly :
    LCurly # getLCurly
    ;
rcurly :
    RCurly # getRCurly
    ;
lbracket :
```

```
    LBracket # getLBracket
    ;
rbracket :
    RBracket # getRBracket
    ;
assign :
    Assign # getAssign
    ;
lineend:
    LineEnd # getLineEnd
    ;
comma:
    Comma # getComma
    ;


Ident :
    [a-zA-Z]([a-zA-Z] | '_' | [0-9])*
    ;

IntConstant : '0' | [1-9][0-9]*;
RealConstant : IntConstant('.'([0-9]+))? ;
BooleanConstant : 'true' | 'false' ;

CompOp : '<=' | '>=' | '>' | '<' | '!=' | '==' | '<>' |;
MulDivMod : '*' | '/' | '%';
AddMin : '+' | '-' ;

LParen : '(' ;
RParen : ')' ;
LCurly : '{' ;
RCurly : '}' ;
LBracket : '[' ;
RBracket : ']' ;

Assign : '=' ;

LineEnd : ';' ;

Comma : ',' ;

WS : [' '\t\r\n]+ -> skip ;

SL_COMMENT :   '//' ~[\r\n]* -> skip;
MUL_COMMENT :   '/*' .*? '*/' -> skip;
```

> 其中加下划线的部分，是为了方便实验二中将词法当作语法产生输出，此处可以忽略。

(2)语法分析方法：递归下降法

ANTLR工具通过语法规则生成一个递归下降的语法分析器。

递归下降的语法分析器是一个递归方法的集合，解析过程则是从根开始向叶子递归。

在解析时， 当一个规则存在多个选项时，则需要检查接下来输入的一个或者多个单词来做出预测判断，在规则的多个选项中选择一个，这样的工作由ANTLR自动完成。

2.程序结构说明

ANTLR在它的运行库中提供了两种树的遍历机制。

1. 默认情况下，ANTLR生成一个语法树的监听器来响应遍历语法树触发的事件。监听器是自动按照深度优先完成树的遍历，我们只需完成在每个节点需要触发的各种事件；
2. 而访问器提供了可控的遍历方式，我们自行控制遍历，决定是否调用子结点的访问方法。由于本课题使用了监听器的方式来实现，所以详细阐述监听器的机制。

为了能够遍历语法树并在监听器中触发事件的调用，ANTLR的运行时提供了一个类`ParseTreeWalker`。在语言的应用程序中，我们创建`ParseTreeListener`类的子类来包含应用程序特定的代码。

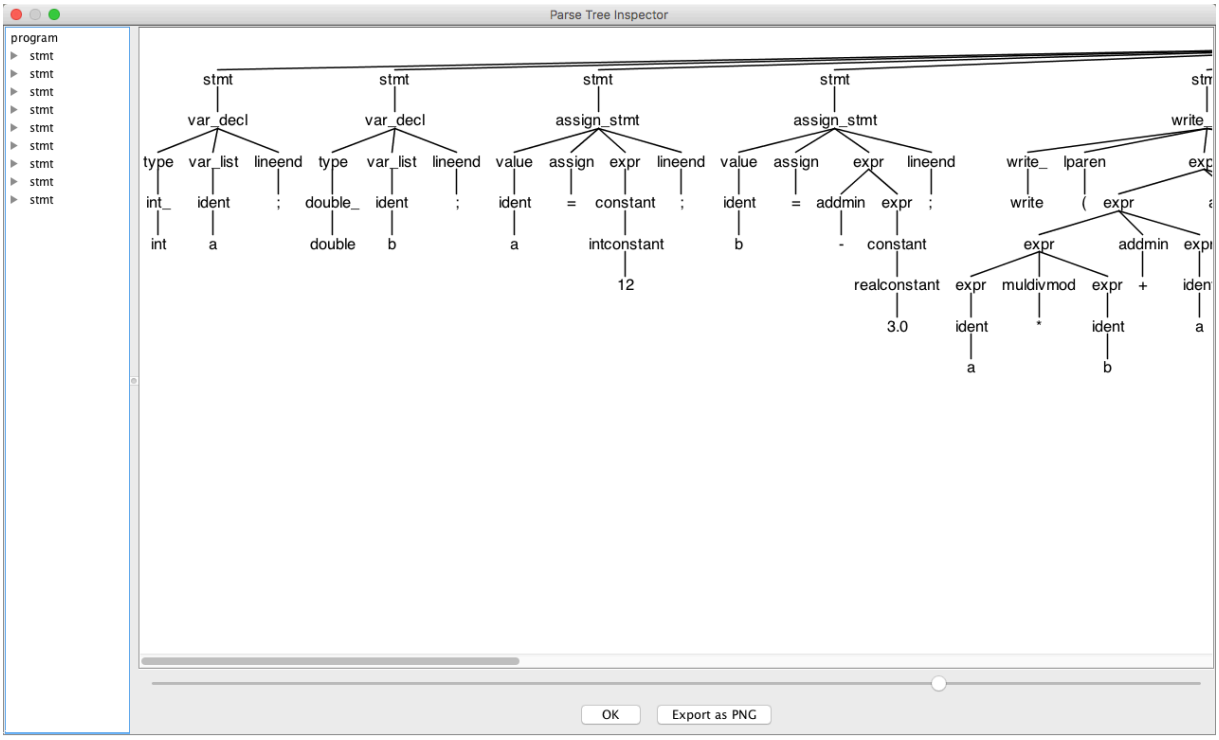
ANTLR为每一个条规则生成一个进入和退出的方法，当遍历进入到这个节点是会触发进入的方法，当遍历完这个节点所有的子节点后，会触发退出的方法。监听器的机制为会自动完成语法树的遍历，无需再写语法树的遍历过程，监听器会提供一个类，包含每个规则的输入输出函数，我们只需继承这个类并复写这些函数即可。

3.程序调试

已经将程序打包为jar包`cmm.jar`，并提供测试文件`test.cmm`。使用方法为

```
$ java -jar cmm.jar test.cmm
```

会自动打印出语法树并显示出来，可以选择保存到png图片格式。



示例测试文件：

```
int a;
double b;
a = 12;
b = -3.0;
write (a*b + a - b);

while (a >= 7) {
    a = a - 1;
}

int c;
c = 4;
/* 声明同时赋值将会在后面实验实现 */

if (c == 4) {
    a = a - 1;
} else {
    a = a + 1;
}
```

> 也可自行编写`.cmm`文件，作为java的第一个参数传入即可。

三、实验总结

通过这次试验，我学习到了通过antlr进行语法分析的过程。同时复习了上学期学习的“递归下降法”并在实践中加深了理解。

这个cmm解释器还有一些不足：

- 报错信息不够完善
- 仅仅实现了最基本的规定语法，有些更好的语法还没有实现

在今后的实验中会抓紧时间实现。