1 编译原理作业之语法分析器

1.1 设计思路

本项目是用Java语言编写的针对于C语言的小型词法分析器。在构建词法分析器的过程中使用了ANTLR工具辅助生成词法分析器生成器。

运行该词法分析器需要安装Java虚拟机。

作业提交分为源代码、jar包(Complier.jar)和报告三部分。

其中jar包可以直接在虚拟机上运行

1.2 ANTLR

ANTLR—Another Tool for Language Recognition，其前身是PCCTS，它为包括Java，C++，C#在内的语言提供了一个通过语法描述来自动构造自定义语言的识别器（recognizer），编译器（parser）和解释器（translator）的框架。ANTLR可以通过断言（Predicate）解决识别冲突；支持动作（Action）和返回值（Return Value）来；更棒的是，它可以根据输入自动生成语法树并可视化的显示出来（这一点我将在下面的例子中演示）。由此，计算机语言的翻译变成了一项普通的任务—在这之前YACC/LEX显得过于学院派，而以LL（k）为基础的ANTLR虽然在效率上还略有不足，但是经过近些年来的升级修改，使得ANTLR足以应付现存的绝大多数应用。感谢Terence Parr博士和他的同事们十几年来的出色工作，他们为编译理论的基础和语言工具的构造做了大量基础性工作，也直接导致了ANTLR的产生。

1.3 语法分析功能

本语法分析器是在上一次作业词法分析器的基础上实现的。

1.3.1 词法分析器实现的功能

- 完成了 **预处理器**，可以处理C语言文件中 # 、//、 /**/ 等符号
- 增加了可识别的符号和关键字的数量

  关键字 **return main case switch true false if else do for while new int float double long char struct**

  符号
- 完成了识别实数和整数的功能
- ANTLR会自动完成词法分析出错处理的功能

1.3.2 语法分析器实现的功能

通过在ANTLR的语法文件中加入限制语法的语句来描述语法分析器的语法。

本语法分析器实现了

- 分析 **for**和 **while**循环
- 分析 **数组变量**
- 分析 **函数定义**
- 分析 **if -else- else if**
- **过滤C语言中的预处理语句**
- 分析 **return 语句**
- **判断加减乘除运算的优先级**

```
grammar CB;      // 定义一个名为Hello的语法，名字与文件名一致


//@header{package com.zetyun.aiops.core.math;}

prog : statlist;


block:'{'(' ')*'}'
     |'{' statlist '}';


statlist:stat
| statlist stat;




stat: expr';'
    |typename? (' ')* ID '=' expr ';'
    |NUM '='expr ';'
    |typename? ID numlist ';'
    |forexpr
    |funtions
    |jump_stat';'
    |declare';'
    |ifexpr ;



ifexpr:IF '('expr')'block
      |IF '('expr')'block WS* (' ')* ELSE IF block
      |IF '('expr')'block WS* (' ')* ELSE block;

forexpr:FOR '('typename? (' ')* ID '=' expr   ';'  expr';' autocCacu ')' block
       |FOR '('    (' ')*    ')'block
       |WHILE'(' (' ')*expr (' ')*')'block
;
//自增与自减
autocCacu:ID'++'
         |ID'--';



funtions:typename (' ')*  (ID|'main') '('formals ')'block

;
//参数表
formals:(' ')*
       |formal
       |formal','formals;

formal:typename(' ')+ ID;
```

```
expr:
|mulexpr(('+'|'-')mulexpr)*
|expr LE expr
|expr GE expr
|expr GT expr
|expr LT expr
;

listype:ID numlist;


numlist:'['']'
|'['NUM']'
|numlist (('[' ']')|('['NUM']'))
;




mulexpr: atom(('*'|'/') atom) *;



atom:'('expr')'
|listype
|NUM
|ID;

//声明语句
declare:typename(' ')* ID numlist
|typename(' ')*idlist;


idlist: ID idlist2;

idlist2:
|','(' ')*ID (' ')*idlist2;



typename:
        |VOID
        |FLOAT
        |INT
        |DOUBLE
        |LONG;

//有关跳转的语句
jump_stat   : 'continue'
            | 'break'
            | 'return'(' ')* expr
            | 'return'
            | 'return'(' ')* NUM ;

NEWLINE:'\r'?'\n'->skip;
NOTHING:(' ')->skip;
WS : [\t\n\r\f]+-> skip;
```

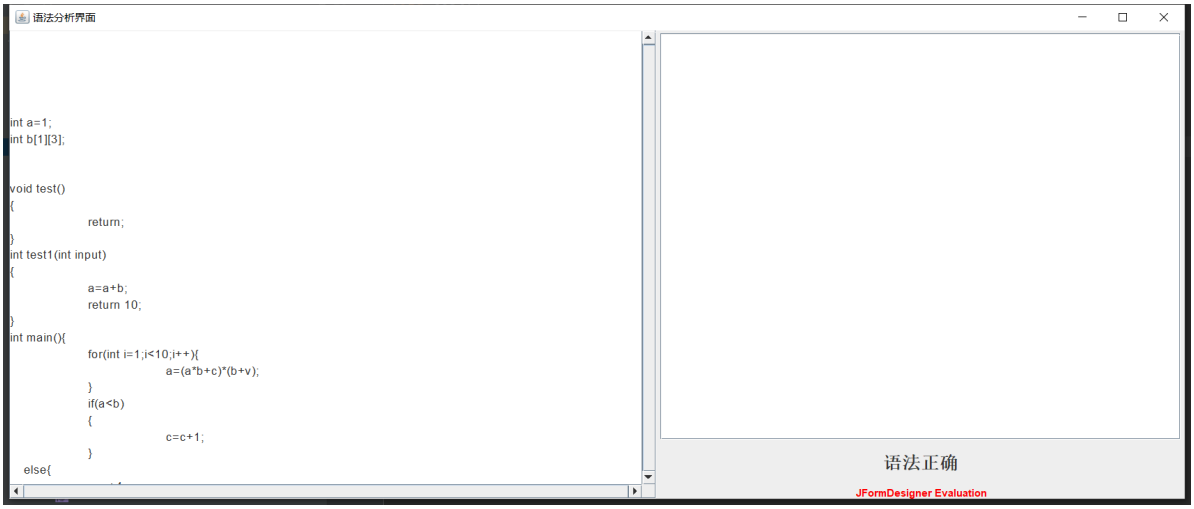
```
COMMENT1:'/*'.*?'*/'->channel(2);
COMMENT2:'//'.*?'\n' ->channel(2);
PRE:'#'.*?'\n'->channel(3);









MUL : '*' ; // assigns token name to '*' used above in grammar
DIV : '/' ;
ADD : '+' ;
SUB : '-' ;

ASSIGN:'=';
LT:'<';
LE:'<=';
GT:'>';
GE:'>=';
AND:'&&';
OR:'||';
XOR:'^';


Semicolon:';';//分号
Colon:':';

Point:'.';
COMMA:',';
LBrac:'(';
RBrac:')';
LBBrac:'{';
RBBrac:'}';
LSB:'[';
RSB:']';


VOID:'void';

STRUCT:'struct';

IF:'if';
ELSE:'else';

FOR:'for';
WHILE:'while';
DO:'do';
SWITCH:'switch';
CASE:'case';

TRUE:'true';
FALSE:'false';
```

```
NEW:'new';
RETURN:'return';
MAIN:'main';

INT:'int';
FLOAT:'float';
DOUBLE:'double';
LONG:'long';
CHAR:'char';


ID : [a-zA-Z][a-zA-Z0-9]*;
NUM : [0-9]+;
REAL_NUM:NUM Point NUM;
```

1.4 运行示例

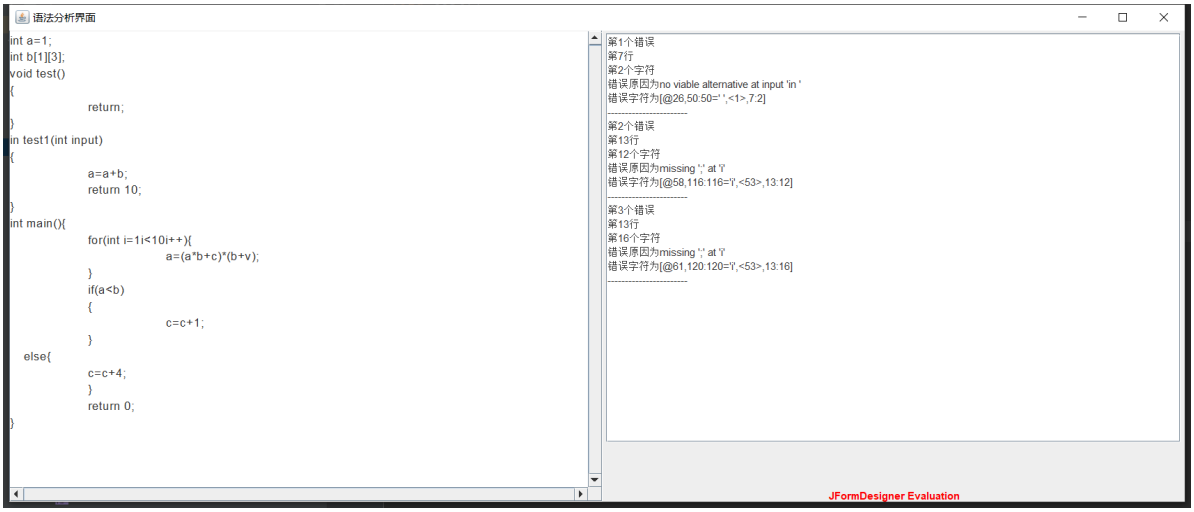
1.4.1 正确示例

```
int a=1;
int b[1][3];
void test()
{
    return;
}
int test1(int input)
{
    a=a+b;
    return 10;
}
int main(){
    for(int i=1;i<10;i++){
        a=(a*b+c)*(b+v);
    }
    if(a<b)
    {
        c=c+1;
    }
    else{
        c=c+4;
    }
    return 0;
}
```



1.4.2 错误示例

```
int a=1;
int b[1][3];
void test()
{
     return;
}
in test1(int input)
{
     a=a+b;
     return 10;
}
int main(){
     for(int i=1i<10i++){
          a=(a*b+c)*(b+v);
     }
     if(a<b)
     {
          c=c+1;
     }
     else{
          c=c+4;
     }
     return 0;
}
```



安装版本>java11的虚拟机,使用命令行输入

```
java -jar Complier.jar
```

就能够运行项目