自动生成语法分析程序（JavaCUP）

下载自动生成工具 JavaCUP

在http://www2.cs.tum.edu/projects/cup/下载得到`java-cup-11b.jar`和`java-cup-11b-runtime.jar`文件，之后存放在项目的`lib`路径下，分别在编译和运行时使用

配置和试用JavaCUP

使用CUP提供的example(http://www2.cs.tum.edu/projects/cup/examples.php)，包括一个`calc.cup`以及`scanner.java`文件

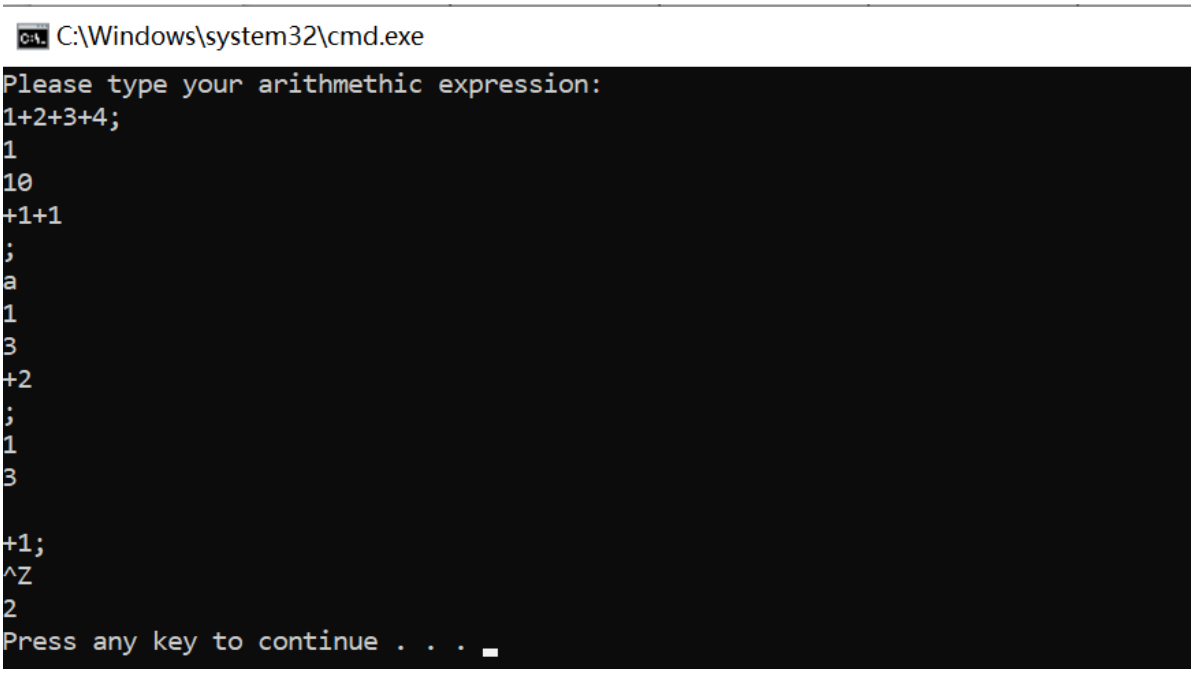
项目结构如下：

```
cup_example
├─bin
├─lib
    ├─java-cup-11b.jar
    ├─java-cup-11b-runtime.jar
├─src
    ├─calc.cup
    ├─scanner.java
    ├─Parser.java   # calc.cup 生成
    ├─sym.java      # calc.cup 生成
├─gen.bat
├─build.bat
├─run.bat
├─yaccgen.pdf
```

使用步骤如下：

1. 使用`java -jar ../lib/java-cup-11b.jar -interface -parser Parser -symbols sym calc.cup`生成语法分析程序`Parser.java`、`sym.java`，
2. 编译`javac -d .\bin -cp .\lib\java-cup-11b-runtime.jar;. .\src\*.java`
3. 运行`java -cp ..\lib\java-cup-11b-runtime.jar;. Main`

能够成功编译运行，虽然在命令行的输出有点顺序问题，不影响正确性



生成 Oberon-0 语法分析和语法制导翻译程序

参照前面的简单示例，实现以下两个文件：`Oberon.cup`,`scanner.java`，再通过`Oberon.cup`生成`Parser.java`、`Symbol.java`，使用实验二的`Oberon.flex`生成`OberonScanner.java`即可。

`scanner.java`**实现:**

在`Oberon.flex`中加入获取token行列的函数

```
%{
    int getLine() {
        return yyline + 1;
    }
    int getColumn() {
        return yycolumn + 1;
    }
%}
```

使用生成的`ObersonScanner`类进行词法分析，可使用数组记录相应信息：token类型，token值，行列值

因为使用JavaCUP进行语法分析需要将输入转为`java_cup.runtime.Symbol`类型，因此通过遍历词法分析过程记录的token，使用`SymbolFactory`生成`Symbol`，最后同样使用`next_token()`函数传出。

`Oberon.cup`**文件实现:**

基本要求为语法分析以及画出过程之间的调用图。

语法分析：给定的BNF中，[]表示该语句可以出现0或1次，{}表示语句出现0或多次，可进行如下改写：

```
A = [a]; -> A = a_block; a_block = a | ;
A = {a}; -> A = a_block; a_block = a_block a | ;
```

在以上改写规则下，将BNF改写成以下.cup文件格式，其中包括了语义分析内容，即画出调用图的内容。画出调用图需要三个步骤：1. 归约一个procedure时`addProcedure`；2. 使用数组，发现过程调用时记录调用语句；3. 在归约一个procedure的同时，检查当前procedure中是否存在调用其他过程，使用`addCallSite`和`addEdge`。

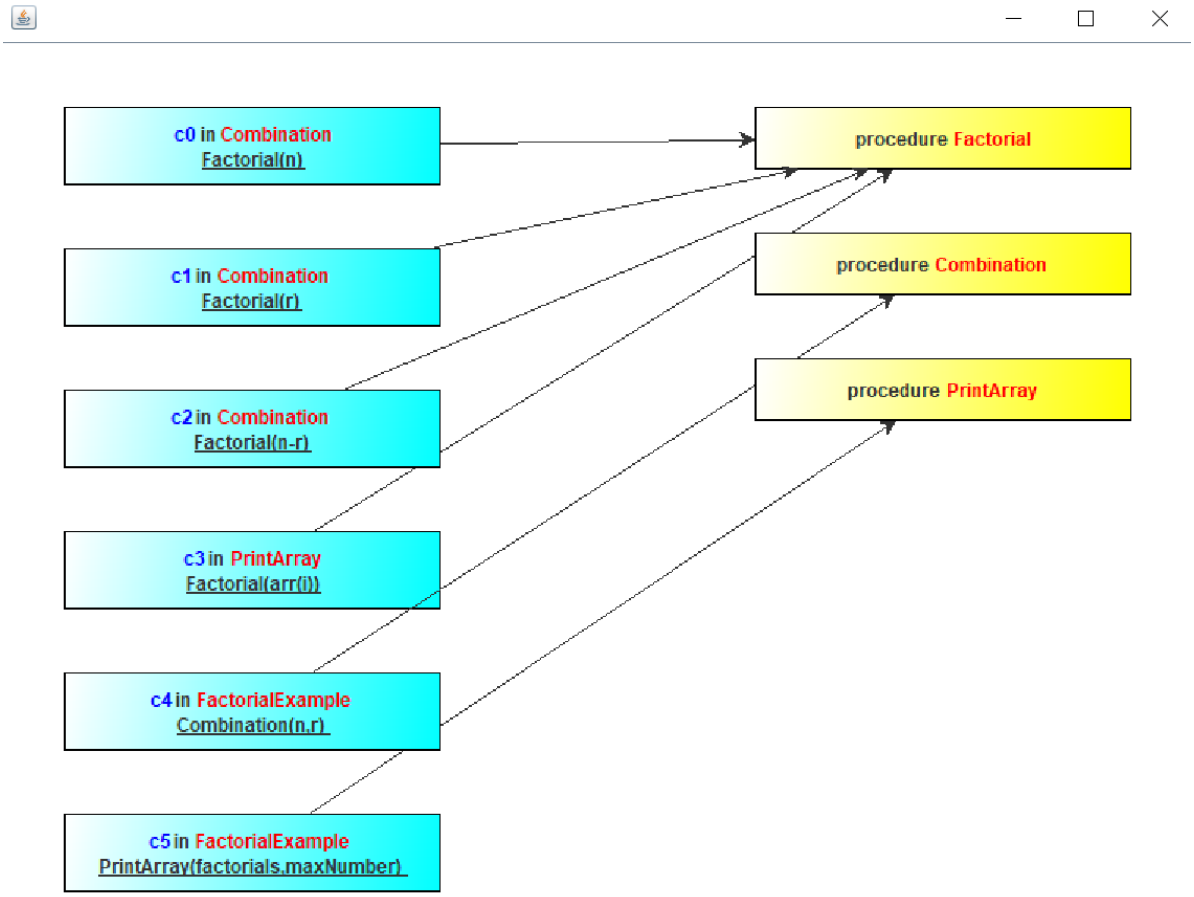
每个序号对应给定的原BNF产生式

```
// 1
modulesBlock    ::= MODULE IDENTIFIER:e SEMI declarations
beginStatementSequenceBlock END IDENTIFIER DOT
                    {:dealCall(e);graph.show(); :};
beginStatementSequenceBlock  ::= BEGIN statementSequence | ;
// 2
declarations    ::= constBlock typeBlock varBlocks procedureDeclarationBlock;
constBlock  ::= CONST identifierExpressionBlock | ;
identifierExpressionBlock   ::= identifierExpressionBlock identifierExpression |
;
identifierExpression    ::= IDENTIFIER EQUAL expression SEMI ;
typeBlock   ::= TYPE identifierTypeBlock | ;
identifierTypeBlock ::= identifierTypeBlock identifierType | ;
identifierType  ::= IDENTIFIER EQUAL types SEMI ;
varBlocks    ::= VAR identifierListTypeBlock | ;
identifierListTypeBlock ::= identifierListTypeBlock identifierListType | ;
identifierListType  ::= identifierList COLON types SEMI ;
procedureDeclarationBlock   ::= procedureDeclarationBlock:e1
procedureDeclarations:e2 | ;
procedureDeclarations   ::= procedureDeclaration SEMI ;
// 3
procedureDeclaration    ::= procedureHeading:e SEMI procedureBody
                        {: graph.addProcedure(e, e);dealCall(e); :};
// 4
procedureBody   ::= declarations beginStatementSequenceBlock END IDENTIFIER;
// 5
procedureHeading    ::= PROCEDURE IDENTIFIER:e formalParametersBlock
                        {: RESULT = e; :};
formalParametersBlock   ::= formalParameters | ;
// 6
formalParameters    ::= fpSectionBlock RIGHTPAR {: if(true) throw new
MissingLeftParenthesisException(); :}
                      | LEFTPAR fpSectionBlock RIGHTPAR ;
fpSectionBlock  ::= fpSection semiFpSectionsBlock | ;
semiFpSectionsBlock  ::= semiFpSectionsBlock semiFpSection | ;
semiFpSection   ::= SEMI fpSection;
// 7
fpSection   ::= varBlock identifierList COLON types ;
varBlock    ::= VAR | ;
// 8
types   ::= IDENTIFIER | arrayType | recordType | INT | BOOL ;
// 9
recordType  ::= RECORD fieldListBlock semiFieldListBlock END;
semiFieldListBlock  ::= semiFieldListBlock semiFieldList |  ;
semiFieldList   ::= SEMI fieldListBlock ;
// 10
fieldListBlock  ::= fieldList |  ;
fieldList   ::= identifierList COLON types;
// 11
```

```
arrayType   ::= ARRAY expression OF types ;
// 12
identifierList  ::= IDENTIFIER commaIdentifierBlock ;
commaIdentifierBlock    ::= commaIdentifierBlock commaIdentifier  |  ;
commaIdentifier ::= COMMA IDENTIFIER  ;
// 13
statementSequence   ::= statementBlock semiStatementBlock ;
semiStatementBlock ::= semiStatementBlock semiStatement | ;
semiStatement  ::= SEMI statementBlock ;
// 14
statementBlock  ::= statement | ;
statement   ::= assignment | procedureCall | ifStatement | whileStatement|
readBlock | writeBlock | writelnBlock ;
// 15
whileStatement  ::= WHILE  expression DO statementSequence END  ;
// 16
ifStatement ::= IF expression THEN statementSequence elsifBlock elseBlock END  ;
elsifBlock ::= elsifBlock elsifStatement |  ;
elsifStatement  ::= ELSIF expression THEN statementSequence ;
elseBlock   ::= elseStatement | ;
elseStatement   ::= ELSE statementSequence ;
// 17
procedureCall   ::= IDENTIFIER:e1 actualParametersBlock:e2
{:addCall(e1,e1+e2);:};
actualParametersBlock   ::= actualParameters:e {: RESULT = e; :} |  {: RESULT =
""; :};
// 18
actualParameters    ::= LEFTPAR expressionBlock {: if (true) throw new
MissingRightParenthesisException(); :}
                      | LEFTPAR expressionBlock:e RIGHTPAR {: RESULT = "(" + e +
")"; :} ;
expressionBlock ::= expressions:e {: RESULT = e; :}  |  {: RESULT = ""; :};
expressions ::= expression:e1 commaExpressionBlocks:e2 {: RESULT = e1 + e2; :} ;
commaExpressionBlocks   ::= commaExpressionBlocks:e1 commaExpressionBlock:e2 {:
RESULT = e1 + e2; :} | {: RESULT = ""; :} ;
commaExpressionBlock    ::= COMMA expression:e {: RESULT = "," + e; :} ;
// 19
assignment  ::= IDENTIFIER selectorBlock COLONEQ expression;
// 20
expression  ::= usimpleExpression:e1 EQUAL usimpleExpression:e2 {: RESULT = e1 +
"=" + e2; :}
              | usimpleExpression:e1 NOTEQUAL usimpleExpression:e2 {: RESULT = e1
+ "#" + e2; :}
              | usimpleExpression:e1 LESS usimpleExpression:e2 {: RESULT = e1 + "
<" + e2; :}
              | usimpleExpression:e1 LEQ usimpleExpression:e2 {: RESULT = e1 + "
<=" + e2; :}
              | usimpleExpression:e1 GREAT usimpleExpression:e2 {: RESULT = e1 +
">" + e2; :}
              | usimpleExpression:e1 GEQ usimpleExpression:e2 {: RESULT = e1 +
">=" + e2; :}
              | usimpleExpression:e {: RESULT = e; :};
// 20
usimpleExpression   ::= simpleExpression:e {: RESULT = e; :}
                      | ADD simpleExpression:e {: RESULT = "+" + e; :}
```

```
                        %prec UADD
                      | MINUS simpleExpression:e {: RESULT = "-" + e;:}
                        %prec UMINUS;
simpleExpression   ::= term:e {: RESULT = e; :}
                      | simpleExpression ADD {:if (true) throw new
MissingOperandException(); :}
                      | simpleExpression MINUS {:if (true) throw new
MissingOperandException(); :}
                      | simpleExpression OR {:if (true) throw new
MissingOperandException(); :}
                      | simpleExpression:e1 ADD simpleExpression:e2 {: RESULT =
e1 + "+" + e2; :}
                      | simpleExpression:e1 MINUS simpleExpression:e2 {: RESULT =
e1 + "-" + e2; :}
                      | simpleExpression:e1 OR simpleExpression:e2 {: RESULT = e1
+ "OR" + e2; :};
// 21
term    ::= term MUL {: if (true) throw new MissingOperandException(); :}
          | term DIV {: if (true) throw new MissingOperandException(); :}
          | term MOD {: if (true) throw new MissingOperandException(); :}
          | term AND {: if (true) throw new MissingOperandException(); :}
          | term:e1 MUL term:e2 {:  RESULT = e1 + "MUL" + e2; :}
          | term:e1 DIV term:e2 {:  RESULT = e1 + "DIV" + e2; :}
          | term:e1 MOD term:e2 {:  RESULT = e1 + "MOD" + e2; :}
          | term:e1 AND term:e2 {:  RESULT = e1 + "&" + e2; :}
          | factor factor {: if (true) throw  new MissingOperatorException(); :}
          | factor:e {:  RESULT = e; :} ;
// 22
factor  ::= IDENTIFIER:e1 selectorBlock:e2 {: RESULT = e1 + e2; :}
          | NUMBER:e {: RESULT = e; :}
          | LEFTPAR expression:e RIGHTPAR {: RESULT = "(" + e + ")"; :}
          | NOT factor:e  {: RESULT = "~" + e; :};
// 23
selectorBlock   ::= selectorBlock:e1 selector:e2 {: RESULT = e1 + e2; :}
                  | {: RESULT = ""; :};
selector    ::= DOT IDENTIFIER:e {: RESULT = "." + e; :} | LEFTMIDPAR
expression:e RIGHTMIDPAR {: RESULT = "(" + e + ")";:} ;
// other
readBlock ::= READ actualParametersBlock ;
writeBlock  ::= WRITE actualParametersBlock ;
writelnBlock::= WRITELN actualParametersBlock ;
```

实现`Oberon.cup`和`scanner.java`两个文件后，生成其他java文件并编译运行正确的示例程序(依次运行`gen.bat` `build.bat` `run.bat`)，即可得到下图:



运行具有语法错误的测试文件`test.bat`:

```
Running Testcase 007: MissingRightParenthesisException
================================================
Lexical analysis done. With 0 lexical error
Error happen at line 26, column 9.
exceptions.MissingRightParenthesisException: Right parenthesis ')' is expected.
        at Parser$CUP$Parser$actions.CUP$Parser$do_action_part00000000(Parser.java:1307)
        at Parser$CUP$Parser$actions.CUP$Parser$do_action(Parser.java:1886)
        at Parser.do_action(Parser.java:470)
        at java_cup.runtime.lr_parser.parse(lr_parser.java:699)
        at Main.main(scanner.java:21)
================================================
Press any key to continue . . .

C:\Users\asus\Desktop\大三下\编译原理\lab_3\21307347陈欣宇\ex3>call test008.bat
Running Testcase 008: MissingLeftParenthesisException
================================================
Lexical analysis done. With 0 lexical error
Error happen at line 16, column 31.
exceptions.MissingLeftParenthesisException: Left parenthesis '(' is expected.
        at Parser$CUP$Parser$actions.CUP$Parser$do_action_part00000000(Parser.java:839)
        at Parser$CUP$Parser$actions.CUP$Parser$do_action(Parser.java:1886)
        at Parser.do_action(Parser.java:470)
        at java_cup.runtime.lr_parser.parse(lr_parser.java:699)
        at Main.main(scanner.java:21)
================================================
Press any key to continue . . .
```

```
C:\Users\asus\Desktop\大三下\编译原理\lab_3\21307347陈欣宇\ex3>call test009.bat
Running Testcase 009: MissingOperatorException
==========================================
Lexical analysis done. With 0 lexical error
Error happen at line 24, column 13.
exceptions.MissingOperatorException: An operator is expected.
        at Parser$CUP$Parser$actions.CUP$Parser$do_action_part00000000(Parser.java:1727)
        at Parser$CUP$Parser$actions.CUP$Parser$do_action(Parser.java:1886)
        at Parser.do_action(Parser.java:470)
        at java_cup.runtime.lr_parser.parse(lr_parser.java:699)
        at Main.main(scanner.java:21)
==========================================
Press any key to continue . . .

C:\Users\asus\Desktop\大三下\编译原理\lab_3\21307347陈欣宇\ex3>call test010.bat
Running Testcase 010: MissingOperandException
==========================================
Lexical analysis done. With 0 lexical error
Error happen at line 24, column 12.
exceptions.MissingOperandException: An operand is expected.
        at Parser$CUP$Parser$actions.CUP$Parser$do_action_part00000000(Parser.java:1559)
        at Parser$CUP$Parser$actions.CUP$Parser$do_action(Parser.java:1886)
        at Parser.do_action(Parser.java:470)
        at java_cup.runtime.lr_parser.parse(lr_parser.java:699)
        at Main.main(scanner.java:21)
==========================================
Press any key to continue . . .
```

讨论不同生成工具的差异

1. JavaCUP 和 GNUBison，主要讨论这两种软件工具接收输入源文件时，在语法规则定义方面存在的差异 。

- **语法规则定义**：JavaCUP使用BNF风格的语法规则定义；GNU Bison也使用BNF风格的语法规则定义，但语法规则部分与词法规则部分在同一个文件中。
- **优先级和结合性**：JavaCUP使用`precedence`关键字定义运算符的优先级和结合性；GNU Bison使用`%left`，`%right`等关键字定义运算符的优先级和结合性。
- **动作代码**：JavaCUP使用`{: ... :}`块来包裹。JavaCUP允许在动作代码中直接使用Java代码；GNU Bison则是包含在花括号`{ ... }`内，且Bison中的动作代码通常是C/C++代码

2. 同样基于 Java 语言的分析器生成工具（Parser Generator，即 Compiler Compiler），还有一个名为 JavaCC 的工具。在网上搜索并浏览关于 JavaCC 的相关信息，用最扼要的一两句话指出 JavaCC 与 JavaCUP 的最核心区别。

JavaCC 基于递归下降解析技术进行语法分析，而 JavaCUP 使用 LALR(1) 分析技术