编译原理 PA1_A 实验报告

计 63 肖朝军 2016011302

一、实验目的

1、 建立编程环境、熟悉代码框架、熟悉 Decaf 语言
2、 掌握 Flex 和 BYACC 的具体用法
3、 新增 decaf 语言的部分语法特性

二、实验内容

在本阶段实验中，主要是学会使用 byacc 和 jflex 工具来完成编译过程中的词法分析和语法分析部分。

1、 实验思路

a) 在 Lexer.l 中增加新正规表达式，增加 decaf 语言的操作符与关键字；
b) 在 Parser.y 中增加新的上下文无关文法的产生式，并规定相应运算符的优先级；
c) 修改 SemValue.java，在类中增加新的属性以进行语法分析；
d) 修改 Tree.java，定义新增特性的语法结点，在此文件中定义输出格式等。

2、 实验详细

a) 特性 1：特性 1 是在 decaf 语言中加入支持对象复制的语句。
   i. 内容：新增“scopy”关键字，新增文法产生式，新增 Tree 语法结点，用于表示 “scopy (IDENTIFIER, Expr);”。
   ii. 代码片段

```
OCStmt          :   SCOPY '(' IDENTIFIER ',' Expr ')'
                    {
                        $$.stmt = new Tree.Scopy($3.ident, $5.expr, $1.loc);
                    }
                ;
```

```
public static class Scopy extends Tree {
    public String name;
    public Expr expr;

    public Scopy(String name, Expr expr, Location loc) {...}

    @Override
    public void accept(Visitor v) { v.visitScopy( that: this); }

    @Override
    public void printTo(IndentPrintWriter pw) {...}
}
```

b) 特性 2：特性 2 是为类的定义引入 sealed 属性。
   i. 内容：新增“sealed”关键字，修改了原有的 ClassDef 的文法产生式，使其能够接受“sealed”属性；在 SemValue 中增加 seal(Boolean)属性；修改 Tree.ClassDef 类，在其中加入了 sealed(boolean)属性
   ii. 代码片段

```
ClassDef        :   SealDef CLASS IDENTIFIER ExtendsClause '{' FieldList '}'
                    {
                        $$.cdef = new Tree.ClassDef($1.seal, $3.ident, $4.ident, $6.flist, $1.loc);
                    }
                ;

SealDef         : SEALED
                    {
                        $$.seal = Tree.SEALED;
                    }
                |   /* empty */
                    {
                        $$ = new SemValue();
                        $$.seal = Tree.UNSEALED;
                    }
                ;
```

c) 特性 3：串行条件卫士语句

i. 内容：新增了“|||”运算符；将 IfSubStmt* 改成可接受的上下文无关文法产生式；新增了 Tree.IfSub 语法结点和 Tree.Guard 语法结点；在 SemValue 中新增 List<IfSub> iflist 与 Tree.IfSub ifsub 成员变量。

ii. 代码片段

```
GuardedStmt     :   IF '{' IfSubStmtList '}'
                    {
                        $$.stmt = new Tree.Guard($3.iflist, $1.loc);
                    }
                ;

IfSubStmtList   :   IfSubStmtList IFOR IfSubStmt
                    {
                        $$.iflist.add($3.ifsub);
                    }
                |   IfSubStmt
                    {

                        $$.iflist = new ArrayList<Tree.IfSub>();
                        $$.iflist.add($1.ifsub);
                    }
                |   /* empty */
                    {
                        $$.iflist = new ArrayList<Tree.IfSub>();
                    }
                ;

IfSubStmt       :   Expr ':' Stmt
                    {
                        $$.ifsub = new Tree.IfSub($1.expr, $3.stmt, $1.loc);
                    }
                ;
```

d) 特性 4：支持自动类型推导

i. 内容：新增“var”关键字；为 LValue 新增产生式；新增 Tree.VarStmt(extends LValue)语法结点。

ii. 代码片段

```
LValue          :   Receiver IDENTIFIER
                    {
                        $$.lvalue = new Tree.Ident($1.expr, $2.ident, $2.loc);
                        if ($1.loc == null) {
                            $$.loc = $2.loc;
                        }
                    }
                |   Expr '[' Expr ']'
                    {
                        $$.lvalue = new Tree.Indexed($1.expr, $3.expr, $1.loc);
                    }
                |   VarDef
                ;

VarDef          :   VAR IDENTIFIER
                    {
                        $$.lvalue = new Tree.VarStmt($2.ident, $1.loc);
                    }
                ;
```

e) 特性 5.1：数组常量

i. 内容：新增“ArrayConstant”的产生式，将 Constant*改成可接受的产生式写法；新增 Tree.ArrayConstant（Extends Expr）语法结点。

ii. 代码片段

```
Constant        :   LITERAL
                    {
                        $$.expr = new Tree.Literal($1.typeTag, $1.literal, $1.loc);
                    }
                |   '[' ConstantList ']'
                    {
                        $$.expr = new Tree.ArrayConstant($2.elist, $1.loc);
                    }
                |   '[' ']'
                    {
                        $$.expr = new Tree.ArrayConstant(new ArrayList<Tree.Expr>(), $1.loc);
                    }
                |   NULL
                    {
                        $$.expr = new Null($1.loc);
                    }
                ;

ConstantList    :   ConstantList ',' Constant
                    {
                        $$.elist.add($3.expr);
                    }
                |   Constant
                    {
                        $$.elist = new ArrayList<Tree.Expr>();
                        $$.elist.add($1.expr);
                    }
                ;
```

f) 特性 5.2：数组初始化常量表达式

i. 内容：新增“%%”运算符，为左结合，优先级比“++”高，比大于小于号低；新增 Tree.ArrayConstDoubleMod 语法结点；为 Expr 新增产生式。

ii. 代码片段

```
    |   Expr DOUBLEMOD Expr
        {
            $$.expr = new Tree.ArrayConstDoubleMod($1.expr, $3.expr, $1.loc);
        }
```

g) 特性 5.3：数组拼接表达式

i. 内容：新增“++”运算符，为左结合，优先级比低于“+”“-”“%%”；新增 Tree.ArrayDoublePlus 语法结点；为 Expr 新增产生式。

ii. 代码片段

```
    |   Expr DOUBLEPLUS Expr
        {
            $$.expr = new Tree.ArrayDoublePlus($1.expr, $3.expr, $1.loc);
        }
```

h) 特性 5.4：取子数组表达式

i. 内容：为 Expr 新增产生式；新增 Tree.ArraySubArray 语法结点。

ii. 代码片段

```
    |   Expr '[' Expr ':' Expr ']'
        {
            $$.expr = new Tree.ArraySubArray($1.expr, $3.expr, $5.expr, $1.loc);
        }
```

i) 特性 5.5：数组下标动态访问

i. 内容：新增“default”运算符，其为三元运算符，优先级高于其他一元、二元运算符；新增 Tree.ArrayDefault 语法结点。

ii. 代码片段

```
|   Expr '[' Expr ']' DEFAULT Expr
    {
        $$.expr = new Tree.ArrayDefault($1.expr, $3.expr, $6.expr, $1.loc);
    }
```

j) 特性 5.6：数组 comprehension 表达式

i. 内容：为 Expr 新增产生式；新增 Tree.ArrayComp 语法结点。

ii. 代码片段

```
|   '[' Expr FOR IDENTIFIER IN Expr IF Expr ']'
    {
        $$.expr = new Tree.ArrayComp($2.expr, $4.ident, $6.expr, $8.expr, $1.loc);
    }
|   '[' Expr FOR IDENTIFIER IN Expr ']'
    {
        $$.expr = new Tree.ArrayComp($2.expr, $4.ident, $6.expr, new Tree.Literal(Tree.BOOL, true, $7.loc), $1.loc);
    }
```

k) 特性 5.7：数组迭代语句

i. 内容：新增“foreach”、“in”关键词；新增 ForeachStmt 非终结符，并定义了系列产生式；定义了 Tree.ForeachStmt 语法结点。

ii. 代码片段

```
ForeachStmt     :   FOREACH '(' ForeachType IDENTIFIER IN Expr WHILE Expr ')' Stmt
                    {
                        $$.stmt = new Tree.ForeachStmt($3.type, $4.ident, $6.expr, $8.expr, $10.stmt, $1.loc);
                    }
                |   FOREACH '(' ForeachType IDENTIFIER IN Expr')' Stmt
                    {
                        $$.stmt = new Tree.ForeachStmt($3.type, $4.ident, $6.expr, new Tree.Literal(Tree.BOOL, true, $7.loc), $8.stmt, $1.loc);
                    }
                ;
```

3、 实验中碰到的困难

a) 刚刚初步接触该实验框架时，一头雾水，不知道该如何下手读代码。在充分阅读各个实验文档之后，开始阅读代码中已经实现的一些语法特性。第一个特性相对简单，因此照猫画虎实现了第一个特性之后，便能够逐渐理解代码背后的一些运行机制。

b) 在定义新的运算符时，需要考虑到运算符的优先级，在实现数组的“%%”时，没有注意到运算符优先级的时候，代码能够编译过，但是却始终输出错误，因此必须注意好各个运算符的优先级，否则将出现错误结果。

c) 要注意应该定义无二义的上下文无关文法，有时会为了保证完备性而写出一些会产生二义性的文法，这将导致很多问题。必须保证文法的唯一性。

三、实验总结

这次实验使我大概了解了实验框架，能够理解词法分析和语法分析这两步需要完成的工作，为了之后的实验做好了铺垫。在实验过程中，也慢慢理解了程序高级语言作为一种形式语言，需要一个无二义性文法的重要性。

感谢助教老师的帮助！