

# Decaf PA 1-A

王世因 2016011246

## 任务描述

根据实验讲义上的说明，我需要通过更改以下的文件达到附加的功能：

1. 支持对象复制语句。
2. 引入关键词 `sealed` 修饰 `class`，使其不能被继承。
3. 支持串行条件卫士语句。
4. 支持简单的自动类型推导。
5. 支持若干与一维数组有关的表达式或语句。
6. Python 风格的数组 `comprehension` 表达式。
7. 数组迭代语句。

文件名	含义	说明
Lexer.l	LEX 源程序	你要在此文件中定义正规式，并给出相应的动作。
Parser.y	YACC 源程序	你要在其中加入 Decaf 的语法规则和归约动作
SemValue	文法符号的语义信息	你要根据自己的需要进行适当的修改
<code>ParserHelper</code>	编写 YACC 动作的辅助类	在这里编写 yacc 的动作，然后粘贴到 Parser.y 中
tree/*	抽象语法树的各种结点	你要在此文件中定义实验新增特性的语法节点
Driver	Decaf 编译器入口	调试时可以修改
utils/*	辅助的工具类	可以增加，不要修改原来的部分

## 具体实现

### 1. 修改bug

Lexer.l

添加this的保留字：

```
"this"          { return keyword(Parser.THIS); }
```

tree.java

有一个明显的bug在于临近的两个生命名称定义不是相连的：

```

/**
 * Labelled statements, of type Labelled.
 */
public static final int LABELLED = FORLOOP + 1;

/**
 * Synchronized statements, of type Synchronized.
 */
public static final int SYNCHRONIZED = CASE + 1;

```

改正过这个以后，代码就可以初步地跑起来力，程序通过了

`error*.decaf`、`fibonacci.decaf`、`nqueues.decaf` 和 `test*.decaf`。

## 2. 增加sealed

我模仿了原有代码中对于static function的部分，修改了ClassDef的部分代码，照猫画虎。

```

ClassDef:SEALED CLASS IDENTIFIER ExtendsClause '{' FieldList '}'
{
    $$.$cdef = new Tree.ClassDef(true, $3.ident, $4.ident, $6.flist,
    $1.loc);
}
|CLASS IDENTIFIER ExtendsClause '{' FieldList '}'
{
    $$.$cdef = new Tree.ClassDef(false, $2.ident, $3.ident,
    $5.flist, $1.loc);
}
;

```

```

public static class ClassDef extends Tree {
    public String name;
    public String parent;
    public List<Tree> fields;
    public boolean sealed;

    public ClassDef(boolean sealed, String name, String parent, List<Tree>
    fields,
        Location loc) {
        super(CLASSDEF, loc);
        this.name = name;
        this.parent = parent;
        this.fields = fields;
        this.sealed = sealed;
    }

    @Override
    public void accept(Visitor v) {
        v.visitClassDef(this);
    }
}

```

```

    }

    @Override
    public void printTo(IndentPrintWriter pw) {
        if(sealed){
            pw.print("sealed ");
        }
        pw.println("class " + name + " "
            + (parent != null ? parent : "<empty>"));
        pw.incIndent();
        for (Tree f : fields) {
            f.printTo(pw);
        }
        pw.decIndent();
    }
}

```

### 3. 增加scopy

这个比较简单，照着实验说明上的写就好，为了以后的使用，注意要把scope的两个参数的类型写好。

```

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

```

```

public static class Scopy extends Tree{
    public String identifier;
    public Expr expr;

    public Scopy(String identifier, Expr expr, Location loc){
        super(SCOPY, loc);
        this.identifier = identifier;
        this.expr = expr;
    }

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

    @Override
    public void printTo(IndentPrintWriter pw) {
        pw.println("scopy");
        pw.incIndent();
        pw.println(identifier);
    }
}

```

```

        expr.printTo(pw);
        pw.decIndent();
    }
}

```

## 4. 增加var

我第一反应是按照对 `sealed class` 的处理，但是看到实验文档后觉得还是要单独设置一个 `class` 便于日后修改调用。

```

/**
 * A var identifier
 */
public static class IdentVar extends LValue {

    public String name;
    public boolean isDefined;

    public IdentVar(String name, Location loc) {
        super(IDENTVAR, loc);
        this.name = name;
    }

    @Override
    public void accept(Visitor v) {
        v.visitIdentVar(this);
    }

    @Override
    public void printTo(IndentPrintWriter pw) {
        pw.println("var " + name);
    }
}

```

## 5. 条件卫士

我仿照 `StmtBlock` 的写法，给Guarded的内部的条件列表做打印

```

GuardedStmt:IF '{' IfBranchG IfStmtG}'
    {
        $3.slist.add($4.stmt);
        $$stmt = new Tree.Guard($3.slist, $1.loc);
    }
;

IfBranchG:IfBranchG IfStmtG DIVIDER
    {
        $$slist.add($2.stmt);
    }
;

```

```

    }
    | /* empty */
    {
        $$ = new SemValue();
        $$.slist = new ArrayList<Tree>();
    }
;

IfStmtG:Expr ':' Stmt
{
    $$stmt = new Tree.IfG($1.expr, $3.stmt, $1.loc);
}
;

```

完成了前四个后，我逐渐熟悉了实验环境，条件卫士写的比较轻松。

## 6. 自动类型推导var

```

LValue : VAR IDENTIFIER
{
    $$lvalue = new Tree.IdentVar($2.ident, $2.loc);
    if ($1.loc == null) {
        $$loc = $2.loc;
    }
}
| 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);
}
;

```

仿照 `VarDef` 写的自动类型推导的递归逻辑。

## 7. 数组

相比其他的任务，这个最为繁琐，我零零碎碎折腾了大半天的时间。因为编译器的自动纠错功能有限，在编译失败的时候我需要自己找错误，我不定时地保存下阶段性成果到Github的private repo，有的时候查不出错误，我就直接revert到前一个版本。

```

/**
 * Array
 */

```

```

public static class Array extends Expr{
    public List<Expr> block;

    public Array(List<Expr> block, Location loc) {
        super(ARRAY, loc);
        this.block = block;
    }

    @Override
    public void accept(Visitor v) {v.visitArray(this); }

    @Override
    public void printTo(IndentPrintWriter pw) {
        pw.println("array const");
        pw.incIndent();
        if(block != null){
            for (Expr s : block) {
                s.printTo(pw);
            }
        }
        else{
            pw.println("<empty>");
        }
        pw.decIndent();
    }
}

public static class ArrayConcat extends Expr{
    public Expr e1, e2;

    public ArrayConcat(Expr e1, Expr e2, Location loc) {
        super(ARRAYCONCAT, loc);
        this.e1 = e1;
        this.e2 = e2;
    }

    @Override
    public void accept(Visitor v) {v.visitArrayConcat(this); }

    @Override
    public void printTo(IndentPrintWriter pw) {
        pw.println("array concat");
        pw.incIndent();
        e1.printTo(pw);
        e2.printTo(pw);
        pw.decIndent();
    }
}

```

```

public static class ArrayComp extends Expr{
    public boolean iff;
    public String ident;
    public Expr inbrunch, ifbrunch, output;

    public ArrayComp(boolean iff, Expr output, String ident, Expr
inbrunch, Expr ifbrunch, Location loc) {
        super(ARRAYCOMP, loc);
        this.iff = iff;
        this.ident = ident;
        this.inbrunch = inbrunch;
        this.ifbrunch = ifbrunch;
        this.output = output;
    }

    @Override
    public void accept(Visitor v) {v.visitArrayComp(this); }

    @Override
    public void printTo(IndentPrintWriter pw) {
        pw.println("array comp");
        pw.incIndent();
        pw.println("varbind " + ident);
        inbrunch.printTo(pw);
        if(iff){
            ifbrunch.printTo(pw);
        }
        else{
            pw.println("boolconst true");
        }
        output.printTo(pw);
        pw.decIndent();
    }
}

public static class ArrayFor extends Tree{
    public LValue e1;
    public Tree e2;
    public Expr ident, j;
    public boolean judge;

    public ArrayFor(boolean judege, LValue e1, Expr ident, Tree e2,
Expr j, Location loc) {
        super(ARRAYFOR, loc);
        this.e1 = e1;
        this.e2 = e2;
        this.ident = ident;
        this.judge = judege;
    }
}

```

```

        this.j = j;
    }

    @Override
    public void accept(Visitor v) {v.visitArrayFor(this); }

    @Override
    public void printTo(IndentPrintWriter pw) {
        pw.println("foreach");
        pw.incIndent();
        e1.printTo(pw);

        ident.printTo(pw);

        if(judge){
            j.printTo(pw);
        }
        else{
            pw.println("boolconst true");
        }
        if(e2!=null){
            e2.printTo(pw);
        }
        pw.decIndent();
    }
}

public static class BoundVar extends LValue {

    public String name;
    public TypeLiteral type;

    public BoundVar(TypeLiteral type, String name, Location loc) {
        super(BOUNDVAR, loc);
        this.name = name;
        this.type = type;
    }

    @Override
    public void accept(Visitor v) {
        v.visitBoundVar(this);
    }

    @Override
    public void printTo(IndentPrintWriter pw) {
        pw.print("varbind " + name + " ");
        if(type==null){
            pw.println("var");
        }
    }
}

```



```

        else{
            type.printTo(pw);
            pw.println();
        }
    }
}

public static class ArrayInit extends Expr{
    public Expr e1, e2;

    public ArrayInit(Expr e1, Expr e2, Location loc) {
        super(ARRAYCONCAT, loc);
        this.e1 = e1;
        this.e2 = e2;
    }

    @Override
    public void accept(Visitor v) {v.visitArrayInit(this); }

    @Override
    public void printTo(IndentPrintWriter pw) {
        pw.println("array repeat");
        pw.incIndent();
        e1.printTo(pw);
        e2.printTo(pw);
        pw.decIndent();
    }
}

public static class ArrayRef extends Expr{
    public Expr e1, e2, e3;

    public ArrayRef(Expr e1, Expr e2, Expr e3, Location loc) {
        super(ARRAYREF, loc);
        this.e1 = e1;
        this.e2 = e2;
        this.e3 = e3;
    }

    @Override
    public void accept(Visitor v) {v.visitArrayRef(this); }

    @Override
    public void printTo(IndentPrintWriter pw) {
        pw.println("arrref");
        pw.incIndent();
        e1.printTo(pw);
        pw.println("range");
        pw.incIndent();
    }
}

```

```

        e2.printTo(pw);
        e3.printTo(pw);
        pw.decIndent();
        pw.decIndent();
    }
}

public static class ArrayDefault extends Expr{
    public Expr e1, e2, e3;

    public ArrayDefault(Expr e1, Expr e2, Expr e3, Location loc) {
        super(ARRAYDEFAULT, loc);
        this.e1 = e1;
        this.e2 = e2;
        this.e3 = e3;
    }

    @Override
    public void accept(Visitor v) {v.visitArrayDefault(this); }

    @Override
    public void printTo(IndentPrintWriter pw) {
        pw.println("arrref");
        pw.incIndent();
        e1.printTo(pw);
        e2.printTo(pw);
        pw.println("default");
        pw.incIndent();
        e3.printTo(pw);
        pw.decIndent();
        pw.decIndent();
    }
}

```

## 体会与展望

- 词法分析真是一件繁琐的工作，如果有时间的话我想优化编辑环境，增加自动debug工具。我遇到的bug基本上都是空指针造成的连接问题和传参类型错误，可以建模求解。
- 我这次完全是靠模仿原有代码上手写完，没怎么需要文档，以后应该培养自己从零开始写新语言代码的能力。