# PA 1-A 实验报告

张晨 2017011307

## 实验原理

本次实验中主要涉及以下文件的修改。

- `Decaf.jflex` 负责词法分析，定义关键字。
- `Decaf.jacc` 负责文法分析，用BNF描述语法，定义运算符的优先级。
- `Tree.java` 定义抽象语法树中的各个节点。与`Decaf.jacc`配合，将解析出的具体语法树变成仅含关键信息的抽象语法树。

### 抽象类

1. 在`Decaf.jflex`等中添加`abstract`关键字的声明，在`Decaf.jacc`中添加抽象类/抽象函数的推导。
2. 更改`Tree.java`中函数/类的定义，添加一个关键字表示是否为abstract。将函数的`body`部分设为optional（`Decaf.jacc`中的规则保证了非abstract的函数一定有函数体，故将body设为optional不会导致无函数体的普通函数变合法）。
3. 修复以上更改导致的编译错误。

### 局部类型推断

1. 在`Decaf.jflex`等中添加`var`关键字的声明
2. 模仿`TInt`等增加数据类型`TVar`，方便`Decaf.jacc`中的解析
3. 在`Decaf.jacc`中添加抽象类/抽象函数的规则。为了保证报错时的具体位置正确，我模仿`Var`、`Type`定义了`var`类型的`VarVAR`、`TypeVAR`。由于`var`类型必须初始化，我模仿`Initializer`定义了`InitializerNotNULL`。最后在`SimpleStmt`中使用添加`VarVAR InitializerNotNULL`。
4. 为了能打印出`null`，`LocalVarDef`中不能直接使用`TVar`，而应将`typeLit`改为optional，并在类型是`TVar`的时候将`typeLit`设为null。
5. 修复以上更改导致的编译错误。

### First class function

#### 函数类型

1. 模仿`varList`实现`typeList`。主要包括在`Decaf.jacc`中添加相应规则，在`AbstractParser`中添加类`svType`，在`SemValue`里添加`typeList`。
2. 在`Decaf.jacc`中添加相应推导，在`Tree.java`中添加类型`TLambda`

#### Lambda表达式

1. 在`Decaf.jflex`等中添加`=>`的声明
2. 在`Decaf.jacc`中添加两种lambda表达式的推导。
3. 在`Tree.java`中添加类`Lambda`，继承自表达式`Expr`。因为`Expr`和`Block`都继承自`TreeNode`，所以暂时在类`Lambda`中的相应位置存`TreeNode`，不对两种表达式做具体区分。

### 函数调用

1. 改写Call类型，现在只需要提供`receiver`,`args`,`pos`，其中`receiver`为optional
2. `Decaf.jacc`中更改函数调用的推导
3. 修复导致的编译问题。

# 问题回答

1. AST 结点间是有继承关系的。若结点 `A` 继承了 `B`，那么语法上会不会 `A` 和 `B` 有什么关系？限用 100 字符内一句话说明。

   不一定有关系，抽象语法树上的的子孙关系表示某节点的文法是另一个节点的文法的一部分，但继承关系只能表示逻辑上的A是一种特殊的B，在语法上关系不大。

2. 原有框架是如何解决空悬 else (dangling-else) 问题的？限用 100 字符内说明。

   在移进/归约冲突中，缺省是做移进。故在遇到非最后一个else的时候，会将else算作`stmt`中的一部分。

3. PA1-A 在概念上，如下图所示：

   ```
   作为输入的程序（字符串）
       --> lexer --> 单词流（token stream）
       --> parser --> 具体语法树（CST）
       --> 一通操作 --> 抽象语法树（AST）
   ```

   输入程序 lex 完得到一个终结符序列，然后构建出具体语法树，最后从具体语法树构建抽象语法树。 这个概念模型与框架的实现有什么区别？我们的具体语法树在哪里？限用 120 字符内说明。

   区别：

   1. 框架不显示生成CST，而是直接生成AST。
   2. 框架不生成单词流，每次通过lexer取出一个token。

   具体语法树在decaf.jacc的解析过程中，如果使用了某条规则，就相当于生成了具体语法树中一些点的父子关系。但只能拿到抽象语法树的返回值。