Decaf\_PA3实验报告

计31班 刘智峰 2013011427

【实验内容】

在PA2中，我们已经完成了对输入程序的语义分析工作，此时的输入程序必定是有明确的语义而且不具有不符合语言规范的语义错误的，在接下来的PA3中，我们将对该输入程序进行翻译，把使用带属性修饰的抽象语法树（decorated AST）来表示的输入程序翻译成适合后期处理的另一种中间表示方式。

在Decaf编译器中，为简单起见，我们在AST以外只涉及一种中间表示，这种中间表示叫做三地址码（Three Address Code, TAC），是一种比较接近汇编语言的表示方式。PA3中我们需要把AST表示的程序翻译为跟它在语义上等价的TAC中间表示，并在合适的地方加入诸如检查数组访问越界、数组大小非法等运行时错误的内容。在TAC表示的基础上，经过临时变量的活性分析（PA4内容）、生成汇编代码（在PA4中直接提供）以后，整个编译过程便告完成。

【实验理解】

刚开始做的时候，没有充分理解本次实验新增的实验框架，想像之前两次实验一样通过查看ouput和result文件夹内的样例以及仿照已有的代码，来进行实验。通过这种方法，我只仿照数组大小检查的函数实现了除0检查，其他的就懵了……于是静下心，好好理解了实验框架。在理解的过程中，我询问了计33班的郭志芃同学，他也十分耐心地给我讲解了一遍大概的框架，在此对他表示感谢！

下面以实验框架中自带的visitIf(Tree.if ifStmt)来阐述我的理解。



首先，第三次实验为了维护TAC结构，引入了Temp、Label等数据类型。下面先观察visitIf中的如果没有else的情况，即else if(ifStmt.trueBranch !=null)以下。假设此时的情况为：

If(A){

doSomething;

}

Others；

首先，这里新打了一个表exit，然后用genBeqz函数判断if的条件的值ifStmt.condition.val是否为0，即A是否为0。如果为0，则跳转到刚才打的表exit处，否则就继续往下执行doSomething的内容。那么跳转到exit表，到底跳转到哪呢？可以发现，在if语句结束后，visitIf中有一句genMark(exit)，我理解为，在if结束后的地方给exit打了一个标记，这样如果跳转到exit，即跳转到if执行完后的地方，即此处的Others。

再看visitIf中有else的情况，假设此时的情况为：

If(A){

doSomething;

}else{

doSomething2;

}

Others;

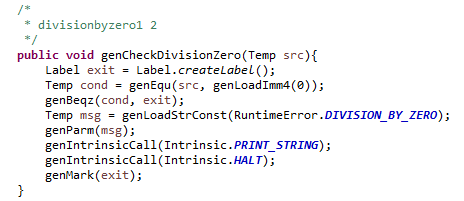
同样的，这里打了一个表falseLabel，然后用genBeqz函数判断if的条件的值ifStmt.condition.val是否为0，即A是否为0。如果为0，则跳到falseLabel。那么falseLabel到底在哪呢？可以发现，在else的语句块被accept之前，即ifStmt.falseBranch.accept(this)之前有genMark(falseLabel)，说明falseLabel的标记在else语句块之前，跳转到falseLabel即跳转去执行else语句。如果不跳转，则顺序执行if内的语句，即doSomething。在这，又打了一个表exit，同时执行完if后，用genBranch跳到了这个表处。同样可以发现，这个表在if-else语句块结束的地方被Mark了，即在Others处被Mark，所以如果if的判断条件正确，则执行完相应的语句后就退出了if-else语句块，继续执行后面的语句。

有了上述理解，对实验的大体框架就有了大概的认识，对于新增语法的支持也就不那么难做了。

【实验一】

1. 实验描述：检测“除零”非法这种运行时错误。
2. 实现思路：仿照判断数组长度是否合法的genCheckArraySize函数来写即可。具体思路就是检测除法和取模操作的第二个操作数是否为0，如果为0则打result中除零错误检测文件中的语句，并跳转到exit表处。如果不是0，则直接跳转到exit表处。即，此函数的关键就是genEqu(src,gemLoadImm(0))。
3. 实现说明：

有了思路，再参考genCheckArraySize函数，实现起来就很简单了。



实现了此函数后，再在原来的genDiv和genMod函数中引用，对第二个操作数进行判断即可。

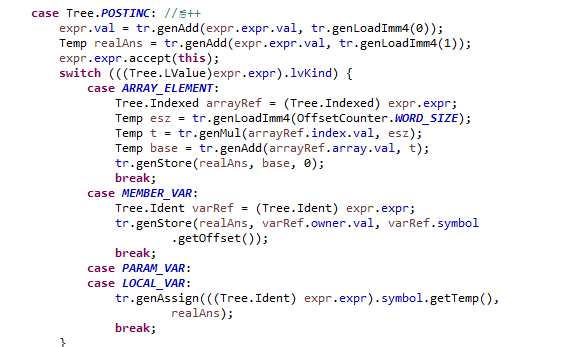
【实验二】

1） 实验描述：实现自增、自减单操作算子的TAC代码生成。自增、自减单操作算子形如i++,++i,i--,--i，其语义解释与C语言中一致。

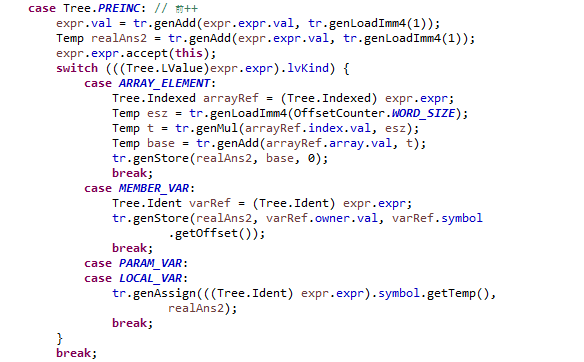
2） 实现思路：++、--运算符，说到底就是一个等式，a++、++a，即a=a+1，唯一的区别就是之后加还是立即加。同样的，a--、--a，即a=a-1，唯一的区别就是之后减还是立即减。因此，只需要仿照原有的处理等式的函数visitAssign就能轻松实现++、--功能。由于在实验1、实验2中我将++、--看做是unary一元运算符，所以我直接在visitUnary中实现了++、--操作。

3） 实现说明：

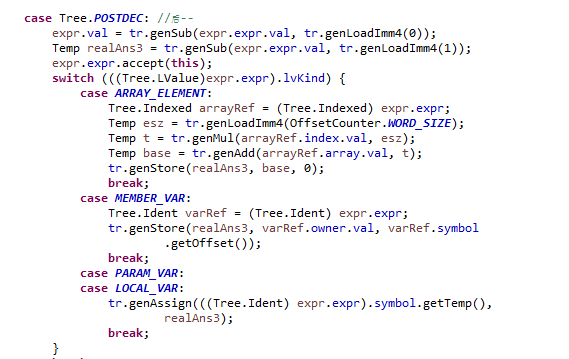
a++：



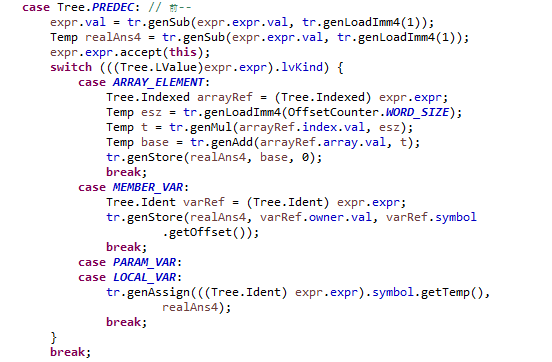
++a：



a--：

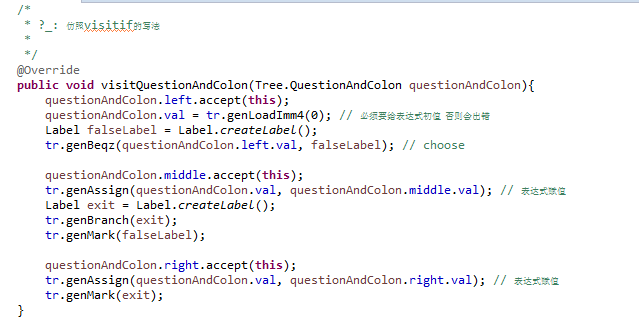


--a：



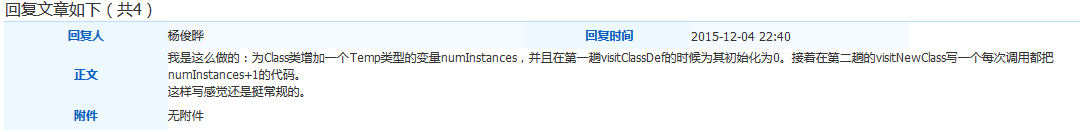
【实验三】

1. 问题描述：实现三操作数算子？：的TAC代码生成。三操作数算子形如A？B：C。
2. 实现思路：“A？B：C”对应规则为Expr1 ? Expr2 : Expr3。此处只需要根据Expr1这个bool式的结果进行跳转即可。新声明一个表falseLabel并在Expr3之前Mark，然后判断Expr1的结果。如果结果为false，则跳到falseLabel执行Expr3，否则继续执行Expr2，并声明一个表exit，在Expr2执行完后用genBranch跳转，并在函数的最后Mark即可。
3. 实现说明：



【实验四】

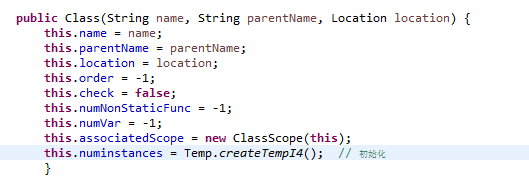
1. 问题描述：实现反射运算numinstances 的TAC代码生成。反射运算numinstances形如 numinstances（A），其语义解释为：计算结果返回类A当前实例对象的个数。
2. 实现思路：numinstances的实现参考了网络学堂的讨论中杨俊晔同学的思路：为Class类增加一个Temp类型的变量numInstances，并且在第一趟visitClassDef的时候为其初始化为0。接着在第二趟的visitNewClass每次调用都把numInstances+1。最后输出numInstances的值即可。



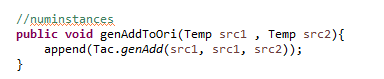
1. 实现说明：

首先在Class.java文件中的Class中新增加了Temp类型的numinstances，并初始化为0。初始化需要调用creatTempI4函数。

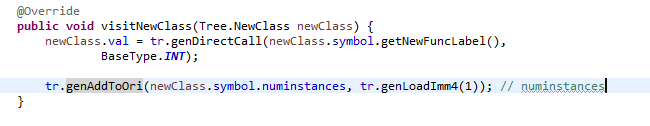




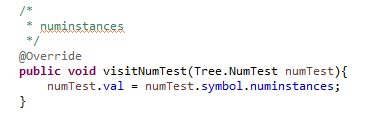
然后在第二次扫描的过程中，每一次visitNewClass，都需要将其中的numinstances加1。由于框架自带的genAdd函数返回的是一个新的Temp，而在统计一个类被new了多少次时需要返回当前这种类而不是新的Temp，所以我新定义了一种加法，返回的Temp为第一个参数。



然后在visitNewClass中调用此函数，对numinstances加1：



最后解析numinstances(C)时，直接输出C的numinstances即可。



【实验五、六】

1. 问题描述：实现串行条件卫士语句和串行循环卫士语句的TAC码生成。语意见实验说明。

2） 实现思路：在理解了visitIf函数后，串行条件卫士语句和串行循环卫士语句就很好写了。对于串行条件卫士语句 if E1:S1 ||| E2:S2 ||| … ||| En:Sn fi，即：

if (E1 == 0) then 跳转到Label1

S1 跳转到Exit

Label1:

If (E2 == 0) then 跳转到Label2

S2 跳转到Exit

Label2

……

if (En == 0) then 跳转到Labeln

Sn 跳转到Exit

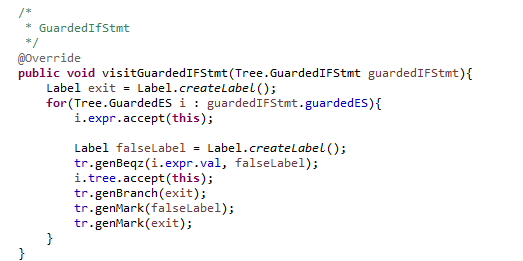
Exit：

串行条件卫士之后的语句

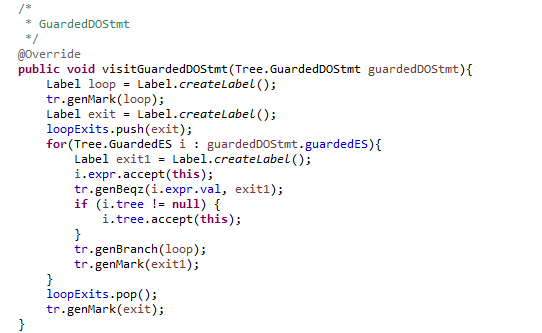
按照这个思路，只需要对visitIf进行稍微修改即可。串行循环卫士语句也类似，只有两点区别：一是需要支持break语句，这点参照visitWhile即可。二是需要引入loop表，同样参照visitWhile即可。

3）实现说明：

串行条件卫士语句：



串行循环卫士语句:



【实验总结】

相比第一、第二阶段的实验，本次实验难度较大，主要是新增加的TAC框架不好理解，不理解框架的话照着框架代码写也写不对。所以理解框架真的很重要，会起到事半功倍的作用。在此再次感谢计33班的郭志芃同学给我讲解实验框架，讲完真是豁然开朗，做起来就很快了！

第一、第二阶段实验都只花了半天时间，而此次实验零零碎碎花了三天。通过本次实验，我对TAC中间代码的生成有了更深层次的理解，对decaf实验也有了更多的认识。感谢老师、助教和郭志芃同学给我的帮助！