# Stage 2 报告

程思翔 2021010761

## 1 实验内容

#### 1.1 step 5

语义分析: 实现了声明语句、赋值语句、标识符的语义分析函数.

在 visitDeclaration 中检查定义域中是否存在定义冲突, 在 visitIdentifier 中需要检查变量是否存在.

```
# frontend/typecheck/namer.py:class Namer
   def visitDeclaration(self, decl: Declaration, ctx: Scope) ->
    None:
       if ctx.lookup(decl.ident.value):
 3
 4
            raise DecafDeclConflictError(decl.ident.value)
        # 构造 VarSymbol 对象, 将其加入符号表, 并设置 decl 的 symbol 属性
 5
        symbol = VarSymbol(decl.ident.value, decl.var t.type)
 6
7
        ctx.declare(symbol)
        decl.setattr("symbol", symbol)
8
9
       if decl.init expr:
            decl.init expr.accept(self, ctx)
10
11
    def visitAssignment(self, expr: Assignment, ctx: Scope) -> None:
12
        # 参考 `visitBinary` 的实现
13
        expr.lhs.accept(self, ctx)
14
        expr.rhs.accept(self, ctx)
15
16
17
   def visitIdentifier(self, ident: Identifier, ctx: Scope) ->
    None:
18
        symbol = ctx.lookup(ident.value)
19
        if not symbol:
20
            raise DecafUndefinedVarError(ident.value)
21
        # 设置 ident 的 symbol 属性
```

**中间代码生成**: 为标识符、声明语句、赋值语句实现了中间代码生成函数. 递归访问每个子节点, 为声明的标识符分配临时寄存器, 并为表达式类型设置返回值 val. 需要注意在赋值时, 左端项需要是左值.

```
# frontend/tacgen/tacgen.py:class TACGen
1
   def visitIdentifier(self, ident: Identifier, mv: TACFuncEmitter)
   -> None:
       # 设置返回值为标识符对应的 temp 寄存器
       ident.setattr("val", ident.getattr("symbol").temp)
 4
5
   def visitDeclaration(self, decl: Declaration, mv:
    TACFuncEmitter) -> None:
7
       decl.getattr("symbol").temp = mv.freshTemp()
       if decl.init expr:
8
           # 对子节点进行 accept
9
           decl.init expr.accept(self, mv)
10
           # 模仿 `visitAssignment` 函数进行赋值
11
12
           decl.setattr(
                "val",
13
   mv.visitAssignment(decl.getattr("symbol").temp,
   decl.init expr.getattr("val"))
14
           )
15
   def visitAssignment(self, expr: Assignment, mv: TACFuncEmitter)
16
   -> None:
       # 对子节点进行 accept
17
       expr.lhs.accept(self, mv)
18
       expr.rhs.accept(self, mv)
19
       # 设置返回值为赋值指令的返回值, 赋值操作更新左值, 左端项是左值 temp
20
21
       expr.setattr(
           "val",
22
   mv.visitAssignment(expr.lhs.getattr("symbol").temp,
   expr.rhs.getattr("val"))
```

```
23
```

目标代码生成: 为赋值语句实现了 visitAssign 函数进行目标代码生成.

```
# backend/riscv/riscvasmemitter.py:class RiscvAsmEmitter
def visitAssign(self, instr: Assign) -> None:
self.seq.append(Riscv.Move(instr.dst, instr.src))
```

## 2 思考题

### 2.1 step 5

1. 我们假定当前栈帧的栈顶地址存储在 sp 寄存器中,请写出一段 **risc-v 汇编 代码**,将栈帧空间扩大 16 字节 (提示1: 栈帧由高地址向低地址延伸;提示2: risc-v 汇编中 addi reg0, reg1, <立即数> 表示将 reg1 的值加上立即数存储到 reg0 中).

答: 汇编代码为:

```
1 addi sp, sp, -16
```

2. 有些语言允许在同一个作用域中多次定义同名的变量, 例如这是一段合法的 Rust 代码 (你不需要精确了解它的含义, 大致理解即可):

```
fn main() {
let a = 0;
let a = f(a);
let a = g(a);
}
```

其中 f(a) 中的 a 是上一行的 let a = 0; 定义的, g(a) 中的 a 是上一行的 let a = f(a);

如果 MiniDecaf 也允许多次定义同名变量, 并规定新的定义会覆盖之前的同名定义, 请问在你的实现中, 需要对定义变量和查找变量的逻辑做怎样的修改 (提示: 如何区分一个作用域中**不同位置**的变量定义?).

答: 在语义分析部分 frontend/typecheck/name.py 中:

- o visitDeclaration 定义变量时,不查询是否有同名变量并抛出同名 异常,因为顺序执行的程序中,新的定义会覆盖之前的定义. 先访问初始 化语句,再访问变量声明,并覆盖原始变量. 这是因为如果存在重名变量 定义,可以先根据变量的初始值计算出其新定义值.
- o visitIdentifier 查找变量无需修改,变量被新定义的变量覆盖后,只需寻找当前作用域中的符号,即是最新定义的变量.