Stage2 实验报告

姓名：丁俊辉

学号：2021011145

step5

实验过程

语义分析阶段：在 `namer.py` 中，我加入了对 `declaration`、`assignment` 和 `Identifier` 的分析。

```
def visitDeclaration(self, decl: Declaration, ctx: Scope) -> None:
    """
    1. Use ctx.lookup to find if a variable with the same name has been declared.
    2. If not, build a new VarSymbol, and put it into the current scope using
ctx.declare.
    3. Set the 'symbol' attribute of decl.
    4. If there is an initial value, visit it.
    """
    if ctx.lookup(decl.ident.value) is not None:
        raise DecafDeclConflictError(decl.ident.name)
    symbol = VarSymbol(decl.ident.value, decl.var_t.type)
    ctx.declare(symbol)
    decl.setattr("symbol", symbol)
    if decl.init_expr is not None:
        decl.init_expr.accept(self, ctx)

def visitAssignment(self, expr: Assignment, ctx: Scope) -> None:
    """
    1. Refer to the implementation of visitBinary.
    """
    expr.lhs.accept(self, ctx)
    expr.rhs.accept(self, ctx)

def visitIdentifier(self, ident: Identifier, ctx: Scope) -> None:
    """
    1. Use ctx.lookup to find the symbol corresponding to ident.
    2. If it has not been declared, raise a DecafUndefinedVarError.
    3. Set the 'symbol' attribute of ident.
    """
    symbol = ctx.lookup(ident.value)
    if symbol is None:
        raise DecafUndefinedVarError(ident.value)
    ident.setattr("symbol", symbol)
```

对于 `declaration`，我会先检查有没有重复声明的符号，如果有则报错，没有则声明该符号。然后，处理声明后可能的初始化情况。

对于 `assignment`，分别检查 lrs 和 rhs。

对于 `Identifier`，检查是否有被声明过，如果没声明则报错。

中间代码生成阶段：在 `tacgen.py` 中，我为 `declaration`、`assignment` 和 `Identifier` 实现了中间代码的生成函数。

```
def visitIdentifier(self, ident: Identifier, mv: TACFuncEmitter) -> None:
    """
    1. Set the 'val' attribute of ident as the temp variable of the 'symbol'
attribute of ident.
    """
    ident.setattr("val", ident.getattr("symbol").temp)

def visitDeclaration(self, decl: Declaration, mv: TACFuncEmitter) -> None:
    """
    1. Get the 'symbol' attribute of decl.
    2. Use mv.freshTemp to get a new temp variable for this symbol.
    3. If the declaration has an initial value, use mv.visitAssignment to set it.
    """
    symbol = decl.getattr("symbol")
    symbol.temp = mv.freshTemp()
    if decl.init_expr is not NULL:
        decl.init_expr.accept(self, mv)
        mv.visitAssignment(symbol.temp, decl.init_expr.getattr("val"))

def visitAssignment(self, expr: Assignment, mv: TACFuncEmitter) -> None:
    """
    1. Visit the right hand side of expr, and get the temp variable of left hand
side.
    2. Use mv.visitAssignment to emit an assignment instruction.
    3. Set the 'val' attribute of expr as the value of assignment instruction.
    """
    expr.rhs.accept(self, mv)
    expr.lhs.accept(self, mv)
    expr.setattr("val", mv.visitAssignment(expr.lhs.getattr("symbol").temp,
expr.rhs.getattr("val")))
```

这些代码支持为声明的标识符分配新的临时寄存器，并在赋值时找到正确的临时寄存器。

目标代码生成阶段：在 `riscvasmemitter.py` 中，我实现了赋值语句的目标代码。

```
def visitAssign(self, instr: Assign) -> None:
    self.seq.append(Riscv.Move(instr.dst, instr.src))
```

思考题

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。）

A：risc-v 汇编代码如下：

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

执行完这条指令后，栈指针（sp）会指向比原来低 16 字节的位置，栈帧空间因此被扩大了 16 字节。

2. 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？）

A：在现有的实现使用了一个集合，每一个符号对应集合中的一个元素，直接在集合中查找来发现缺失/重复。

如果要支持类似 Rust 中 shadowing 的声明特性，我认为应该为每个出现的符号维护一个栈。当出现变量的定义时，就把该变量放到对应符号的栈顶。当变量的生命周期结束的时候，就把变量从对应的栈中弹出。如果要查找变量，就从对应符号的栈顶开始查，查询到第一个符合语法规则的变量，就将其应用。