STAGE-2：变量和语句

计04 何秉翔 2020010944

1. 实验内容

在 stage-2 中，主要完成的部分是 `TAC` 的生成以及 `riscv` 汇编的生成，以及符号表 `namer` 的构建。

1.1 step-5：局部变量与赋值

1.1.1 语义分析

先观察新增加的产生式，注意到出现了一些更改：statement 语句可以导出 declaration，declaration 语句既可以只声明变量，又可以同时定义变量的初值。

```
statement
    : 'return' expression ';'
    | expression? ';'
    | declaration

declaration
    : type Identifier ('=' expression)? ';'

expression
    : assignment

assignment
    : logical_or
    | Identifier '=' expression
```

因为涉及到局部变量的使用，既不能使用未声明的局部变量，也不能重复声明变量，因此需要有一张符号表用于记录已经声明的变量。在生成 `AST` 之后，对 `AST` 做的第一个遍就是遍历语法树构建符号表，检查符号命名和使用的规范。

通过 `tree.py` 的 `accept` 接口，利用继承自 `visitor` 的 `Namer` 类，通过定义上下文环境在 `namer.py` 中进行逐节点的 `visit`，因此按照注释补全 `visitDeclaration`、`visitAssignment` 和 `visitIdentifier`。

- 对于 `visitDeclaration`：
  - 先看这个声明中的 `ident` 是否在符号表中已有同名变量
  - 若有的话则抛出 `DecafDeclConflictError` 异常
  - 若没有则定义一个 `VarSymbol` 对象并 `declare` 出来，记录在符号表中
  - 再将该符号与声明的标识符进行绑定

```
symbol = ctx.findConflict(decl.ident.value)
if symbol == None: # has not been declared
    symbol = VarSymbol(decl.ident.value, decl.var_t.type)
    ctx.declare(symbol)
else:
    raise DecafDeclConflictError(decl.ident.value)
decl.setattr('symbol', symbol)
decl.init_expr.accept(self, ctx)
```

- 对于 `visitIdentifier`：
  - 先在符号表里查一下该标识符

- 若没有，说明还未声明，抛出 `DecafUndefinedVarError` 异常
- 否则把标识符和符号绑定到一起

```
symbol = ctx.lookup(ident.value)
if symbol != None: # has been declared
    ident.setattr('symbol', symbol)
else: # has not been declared
    raise DecafUndefinedVarError(ident.value)
```

- 对于 `visitAssignment`
  - 对于赋值语句，先判断左边的是否是一个左值
  - 若是左值，则左右两边的表达式分别访问
  - 否则抛出 `DecafSyntaxError` 异常

```
if isinstance(expr.lhs, Identifier): # lhs 是左值
    expr.lhs.accept(self, ctx)
    expr.rhs.accept(self, ctx)
else:
    raise DecafSyntaxError('left hand side of assignment is not a left value')
```

1.1.2 中间代码生成

通过 `tree.py` 的 `accept` 接口，利用继承自 `visitor` 的 `TACGen` 类，通过定义上下文环境在 `tacgen.py` 中进行逐节点的 `visit`，因此按照注释补全 `visitDeclaration`、`visitAssignment` 和 `visitIdentifier`。

- 对于 `visitDeclaration`：
  - 首先取得该 `declaration` 在 `namer.py` 中绑定的符号
  - 再为该符号绑定一个临时寄存器
  - 如果 `declaration` 中有赋初始值，则继续访问初值的表达式，并将表达式的值赋值给变量的临时寄存器

```
symbol = decl.getattr('symbol')
symbol.temp = mv.freshTemp()
if decl.init_expr != NULL: # with initial value
    decl.init_expr.accept(self, mv)
    mv.visitAssignment(symbol.temp, decl.init_expr.getattr("val"))
```

- 对于 `visitAssignment`：
  - 首先访问左右两边的表达式，并找到左边表达式（即变量）对应的临时寄存器
  - 将右边的值赋值给左边的寄存器
  - 最后将右边的值赋值给表达式的 `val` 属性

```
expr.lhs.accept(self, mv)
expr.rhs.accept(self, mv)
temp = expr.lhs.getattr("val")
mv.visitAssignment(temp, expr.rhs.getattr("val"))
expr.setattr('val', expr.rhs.getattr("val"))
```

- 对于 `visitIdentifier`：

  直接将标识符对应的符号的临时寄存器的值赋值给标识符的 `val` 属性。

```
ident.setattr('val', ident.getattr('symbol').temp)
```

1.1.3 目标代码生成

在后端文件 `riscvasmemitter.py` 中的 `selectInstr()` 方法里，对于每一条 `TAC` 指令，选择对应的 `riscv` 指令生成 `riscv` 汇编，但我们发现对于 `Assign` 型指令并没有对应的 `visitAssign` 方法，因此在 `RiscvInstrSelector` 类中仿照着利用 `Move` 实现即可：

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

1.2 step-6: if 语句和条件表达式

在本 `step` 中，我们只需仿照 `if` 的实现去实现条件表达式即可。

首先在 `namer.py` 中，我们仿照 `visitIf` 实现 `visitCondExpr` 即可，区别就在于条件表达式一定有 `else` 分支。

```
def visitCondExpr(self, expr: ConditionExpression, ctx: ScopeStack) -> None:
    expr.cond.accept(self, ctx)
    expr.then.accept(self, ctx)
    expr.otherwise.accept(self, ctx)
```

在中间代码生成环节，即 `tacgen.py` 中，我们同样仿照 `visitIf` 实现 `visitCondExpr` ，只是省去 `visitIf` 中没有 `else` 的部分即可。同时注意到条件表达式和 `if` 不同的一点在于，条件表达式是一个表达式，需要有返回值，因此我们需要考虑给表达式 `setattr`，问题就在于 `set` 哪个分支的。

先随便 `set` 一个分支，观察生成的 `TAC`：

```
FUNCTION<main>:
    _T1 = 0
    _T0 = _T1
    _T2 = 1
    if (_T2 == 0) branch _L1
    _T3 = 2
    branch _L2
_L1:
    _T4 = 3
_L2:
    _T0 = _T3
    return _T0
```

注意到程序为两个分支均开了一个临时寄存器，分别为 `_T3` 和 `_T4`，随便 `set` 一个的效果就在于在第 11 行将 `_T3` 还是 `_T4` 的值赋值给 `_T0` 返回。同时也注意到代码中是 `then` 分支比 `otherwise` 分支先 `accept` 的，因此我们可以考虑在 `otherwise` 分支中将已经开出来的 `_T3` 寄存器的值覆盖为 `_T4` 寄存器的值，因为如果程序跑到 `otherwise` 分支里说明 `_T3` 的值肯定是无用的。最后只需统一将 `_T3` 赋值给 `_T0` 即可：

```
expr.cond.accept(self, mv)

skipLabel = mv.freshLabel()
exitLabel = mv.freshLabel()
mv.visitCondBranch(
    tacop.CondBranchOp.BEQ, expr.cond.getattr("val"), skipLabel
)

expr.then.accept(self, mv)
mv.visitBranch(exitLabel)
mv.visitLabel(skipLabel)

```

```
expr.otherwise.accept(self, mv)
mv.visitAssignment(expr.then.getattr("val"), expr.otherwise.getattr("val")) # _T3 = _T4
mv.visitLabel(exitLabel)

expr.setattr("val", expr.then.getattr("val")) # _T0 = _T3
```

2. 思考题

2.1 step-5 思考题

1. **问题**：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。）

   解答：

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

2. **问题**：

   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？）

   解答：

   首先对于 `namer.py`，对于定义同名变量时的检查 `visitDeclaration`，需要将 `DecafDeclConflictError` 的抛出注释掉，允许声明同名变量，在声明到符号表中已存在的变量时，仅仅覆盖声明即可。

   对于 `tacgen.py` 中的 `visitDeclaration` 方法，在调用 `freshTemp` 时，需要判断该声明是否已经有一个临时寄存器，若没有才进行临时寄存器分配，保证定义的同名变量用同一个临时寄存器。

   ```
   if not hasattr(symbol, 'temp'):
       symbol.temp = mv.freshTemp()
   ```

2.2 step-6 思考题

1. **问题**：你使用语言的框架里是如何处理悬吊 else 问题的？请简要描述。

   解答：通过 `--parse` 发现解析是正常解析的，能够解决悬吊 else 的问题，于是考虑 `ply_parser.py`：

```
def p_if_else(p):
    """
    statement_matched : If LParen expression RParen statement_matched Else statement_matched
    statement_unmatched : If LParen expression RParen statement_matched Else statement_unmatched
    """
    p[0] = If(p[3], p[5], p[7])

def p_if(p):
    """
    statement_unmatched : If LParen expression RParen statement
    """
    p[0] = If(p[3], p[5])
```

注意到 `p_if_else` 中，无论是 `matched` 的还是 `unmatched` 的语句，在解析的时候都会将 `if` 的 `then` 分支认为成 `matched` 的语句，即规定 `if` 的 `then` 分支中一定是 `if` 和 `else` 数目相等匹配好的。对于 `if(a) if(b) c=0; else d=0;` `else` 只能跟就近的 `if` 结合，否则，若跟远处的 `if` 结合，那么该远处的 `if` 的 `then` 分支中一定是 `unmatched` 的语句，与规定不一致。

2. **问题**：

   在实验要求的语义规范中，条件表达式存在短路现象。即：

```
int main() {
    int a = 0;
    int b = 1 ? 1 : (a = 2);
    return a;
}
```

会返回 0 而不是 2。如果要求条件表达式不短路，在你的实现中该做何种修改？简述你的思路。

解答：在 `visitCondExpr` 中，只需将 `cond`、`then` 和 `otherwise` 提前 `accept` 即可，之后分支跳转逻辑一样。

```
expr.cond.accept(self, mv)
expr.then.accept(self, mv)
expr.otherwise.accept(self, mv)

skipLabel = mv.freshLabel()
exitLabel = mv.freshLabel()
mv.visitCondBranch(
    tacop.CondBranchOp.BEQ, expr.cond.getattr("val"), skipLabel
)

mv.visitBranch(exitLabel)
mv.visitLabel(skipLabel)

mv.visitAssignment(expr.then.getattr("val"), expr.otherwise.getattr("val"))
mv.visitLabel(exitLabel)

expr.setattr("val", expr.then.getattr("val"))
```