Stage-2 Report

曹伦郗 *2020011020*

实验内容

step5

实现了*Namer.visitDeclaration*，即：

①调用*ctx.findConflict*，根据标识符名称检查所声明的变量是否被重复声明，若重复声明，抛出*DecafDeclConflictError*；

②为声明的新变量创建变量符号，添加到符号表内；

③将该变量符号同*decl*关联；

④若声明时有初始值表达式*init_expr*，访问之。

```
    def visitDeclaration(self, decl: Declaration, ctx: ScopeStack) -> None:
        """
        1. Use ctx.findConflict 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.findConflict(decl.ident.value):
            raise DecafDeclConflictError(decl.ident.value)
        symbol = VarSymbol(decl.ident.value, decl.var_t.type)
        ctx.declare(symbol)
        decl.setattr('symbol', symbol)
        if decl.init_expr != NULL:
            decl.init_expr.accept(self, ctx)
```

实现了*Namer.visitAssignment*，即：

①调用*ctx.lookup*，根据左值的标识符名称检查该变量是否在符号表内（是否已被声明），若未声明，抛出*DecafUndefinedVarError*；

②依次访问左右值表达式。

```
    def visitAssignment(self, expr: Assignment, ctx: ScopeStack) -> None:
        """
        1. Refer to the implementation of visitBinary.
        """
        if not ctx.lookup(expr.lhs.value):
            raise DecafUndefinedVarError(expr.lhs.value)
        expr.lhs.accept(self, ctx)
        expr.rhs.accept(self, ctx)
```

实现了*Namer.visitIdentifier*，即：

①调用*ctx.lookup*，根据该变量的标识符名称从符号表内找到对应变量符号，若不存在（未声明），抛出*DecafUndefinedVarError*；

②将该变量符号同*ident*关联。

```
    def visitIdentifier(self, ident: Identifier, ctx: ScopeStack) -> 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 not symbol:
            raise DecafUndefinedVarError(ident.value)
        ident.setattr('symbol', symbol)
```

实现了*TACGen.visitDeclaration*，即：

①为该*decl*所关联的变量符号分配一个新的临时变量成员*temp*；

②若声明时有初始值表达式*init_expr*，访问之，再调用*mv.visitAssignment*以添加一条赋值指令，将初始值表达式*init_expr*的值赋给该临时变量，并以*mv.visitAssignment*的返回值作为*decl*的值。

```
    def visitDeclaration(self, decl: Declaration, mv: FuncVisitor) -> 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 != NULL:
            decl.init_expr.accept(self, mv)
            decl.setattr('val', mv.visitAssignment(symbol.temp,
decl.init_expr.getattr('val')))
```

实现了*TACGen.visitAssignment*，即：

①访问右值表达式；

②调用*mv.visitAssignment*以添加一条赋值指令，将右值表达式的值赋给左值所关联的变量符号的临时变量成员*temp*，并以*mv.visitAssignment*的返回值作为*expr*的值。

```
    def visitAssignment(self, expr: Assignment, mv: FuncVisitor) -> 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.
        """
        symbol = expr.lhs.getattr('symbol')
        expr.rhs.accept(self, mv)
        expr.setattr('val', mv.visitAssignment(symbol.temp,
expr.rhs.getattr('val')))
```

实现了*TACGen.visitIdentifier*，即：

①以*ident*所关联的变量符号的临时变量*temp*，作为*ident*的值。

```
    def visitIdentifier(self, ident: Identifier, mv: FuncVisitor) -> 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)
```

实现了*RiscvAsmEmitter.RiscvInstrSelector.visitAssign*，即：

①将赋值语句翻译成为*RISC-V*中的*MOVE*指令。

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

step6

实现了*Namer.visitCondExpr*，即：

①依次访问*cond*表达式、*then*表达式、*otherwise*表达式。

```
    def visitCondExpr(self, expr: ConditionExpression, ctx: ScopeStack) -> None:
        """
        1. Refer to the implementation of visitBinary.
        """
        expr.cond.accept(self, ctx)
        expr.then.accept(self, ctx)
        expr.otherwise.accept(self, ctx)
```

实现了*TACGen.visitCondExpr*，即：

①访问*cond*表达式；

②创建两条*label*，分别为*otherwise*情况下*TAC*指令起始处*skipLabel*，条件表达式结束后下一条*TAC*指令起始处*exitLabel*；

③分配一个临时变量*temp*，待为其添加完两种情况下的赋值指令后，再将它作为*expr*的值；

④调用*mv.visitCondBranch*，添加一条条件跳转指令，以跳转至执行*otherwise*情况的一系列指令的入口*skipLabel*；

⑤访问*then*表达式，再调用*mv.visitAssignment*添加一条赋值指令，将*then*表达式的值赋给先前创建的临时变量*temp*，再调用*mv.visitBranch*添加一条跳转指令，以跳转至*exitLabel*；

⑥调用*mv.visitLabel*添加*skipLabel*，再访问*otherwise*表达式，再调用*mv.visitAssignment*添加一条赋值指令，将*otherwise*表达式的值赋给先前创建的临时变量*temp*；

⑦调用*mv.visitLabel*添加*exitLabel*；

⑧将*temp*作为*expr*的值。

```
    def visitCondExpr(self, expr: ConditionExpression, mv: FuncVisitor) -> None:
        """
        1. Refer to the implementation of visitIf and visitBinary.
        """
        expr.cond.accept(self, mv)
        skipLabel = mv.freshLabel()
        exitLabel = mv.freshLabel()
        temp = mv.freshTemp()
        mv.visitCondBranch(tacop.CondBranchOp.BEQ, expr.cond.getattr('val'),
skipLabel)
        expr.then.accept(self, mv)
        mv.visitAssignment(temp, expr.then.getattr('val'))
        mv.visitBranch(exitLabel)
        mv.visitLabel(skipLabel)
        expr.otherwise.accept(self, mv)
        mv.visitAssignment(temp, expr.otherwise.getattr('val'))
        mv.visitLabel(exitLabel)
        expr.setattr('val', temp)
```

思考题

step5

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

2. 要实现同名变量新定义覆盖旧的定义，则在构建符号表遇到声明语句时，不再检查是否重复声明，即*Namer.visitDeclaration*中不再调用*ctx.findConflict*。

   如此，构造的新的*symbol*经过*ctx.declare(symbol)*将直接覆盖原符号表内的旧*symbol*。

step6

1. 在语法分析阶段，*ply_parser.py*中在*p_if_else*和*p_else*的*docstring*中，给出了三条产生式：
   - *statement_matched -> If LParen expression RParen statement_matched Else statement_matched*
   - *statement_unmatched -> If LParen expression RParen statement_matched Else statement_unmatched*
   - *statement_unmatched -> If LParen expression RParen statement*

   这使得如*If...If...Else*的字符串，必须将*Else*跟最近的*If*匹配。原因如下：
   ```

若*Else*跟非最近的*If*匹配，按上述三条产生式中的第一、二条产生式规则，*If*后的执行语句都应当是*statement_matched*。而根据第一条产生式规则，该语句必然是既有*If*又有*Else*的，而实际字符串只有一个*If*。

因此，如*If...If...Else*的字符串，必须将*Else*跟最近的*If*匹配。故*Else*悬吊问题就解决了。

2. 若要实现条件表达式不短路，则应当将对*then*表达式和*otherwise*表达式的访问挪到条件跳转指令之前。即：

```
    def visitCondExpr(self, expr: ConditionExpression, mv: FuncVisitor) ->
None:
        """
        1. Refer to the implementation of visitIf and visitBinary.
        """
        expr.cond.accept(self, mv)
        skipLabel = mv.freshLabel()
        exitLabel = mv.freshLabel()
        temp = mv.freshTemp()
        expr.then.accept(self, mv)
        expr.otherwise.accept(self, mv)
        mv.visitCondBranch(tacop.CondBranchOp.BEQ, expr.cond.getattr('val'),
skipLabel)
        mv.visitAssignment(temp, expr.then.getattr('val'))
        mv.visitBranch(exitLabel)
        mv.visitLabel(skipLabel)
        mv.visitAssignment(temp, expr.otherwise.getattr('val'))
        mv.visitLabel(exitLabel)
        expr.setattr('val', temp)
```