stage-2

计11班 周韧平 2021010699

实验内容

step5 引入了变量，因此需要使用符号表维护 main 函数中定义的变量，即修改 `Namer` 类。首先查找是否已声明了同名变量，如果没有，则创建一个新的，否则报错，然后修改 `decl` 的 symbol 属性，如果后面有赋值操作则继续访问。

```
    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_var = VarSymbol(decl.ident.value,decl.var_t,False)
        ctx.declare(decl_var)
        decl.setattr('symbol',decl_var)
        if decl.init_expr is not None:
            decl.init_expr.accept(self,ctx)
        return      You, 2秒钟前 • Uncommitted changes
        raise NotImplementedError
```

在访问赋值节点时，首先检查左操作数在符号表中是否被定义，否则报错。随后通过调用 `visitBinary` 函数进行访问

```
    def visitAssignment(self, expr: Assignment, ctx: Scope) -> None:
        """
        1. Refer to the implementation of visitBinary.
        """
        symbol = ctx.lookup(expr.lhs.value)
        if symbol is None:
            raise DecafUndefinedVarError(expr.lhs.value)
        expr.lhs.setattr('symbol',symbol)
        self.visitBinary(expr,ctx)
        return
        raise NotImplementedError

```

访问 Identifier 时也要先检查变量是否定义，并设置符号

```
    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)
        return
        raise NotImplementedError

    def visitIntLiteral(self, expr: IntLiteral, ctx: Scope) -> None:
        value = expr.value
        if value > MAX_INT:
            raise DecafBadIntValueError(value)

```

在中端代码中，通过将访问符号的temp设为表达式的返回值实现访问操作

```
    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)
        return
        raise NotImplementedError

```

首先通过mv生成一个临时变量赋给decl的symbol，然后如果有赋值操作，则进一步访问赋值，调用decl的symbol和val进行赋值操作

```
    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.
        """
        decl.getattr('symbol').temp = mv.freshTemp()
        if decl.init_expr is not NULL:
            decl.init_expr.accept(self,mv)
            mv.visitAssignment(decl.getattr('symbol').temp,decl.init_expr.getattr('val'))
        return
        raise NotImplementedError

```

在 `visitAssignment` 中，首先通过访问左右操作数得到其临时变量，然后调用 `visitAssignment` 生成对应的赋值指令

```
    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.lhs.accept(self,mv)      You, 1秒钟前 • Uncommitted changes
        expr.rhs.accept(self,mv)
        temp = mv.visitAssignment(expr.lhs.getattr('symbol').temp,expr.rhs.getattr('val'))
        expr.setattr('val',temp)
        return
        raise NotImplementedError

```

思考题

> 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。）

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

这条指令可以通过将sp减16将栈帧扩大16字节

> 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？）

- 对于变量定义的过程，首先应该取消 `Namer.visitDeclaration` 中对于同名变量重复定义的报错。此外，对于同一作用域中的同名变量，可以考虑通过添加标识符来区分不同位置的变量定义。比如，可以通过变量定义的行号或者使用一个计数器来给同名变量加上唯一的标识符。
- 对于查找变量的过程，应当访问符号表中对应最新定义的变量。比如，如果是使用计数器区分的同名变量，则应该查找计数器值最大的变量。
- 此外，如果在变量定义的同时进行初始化，则应该规定右操作数比左操作数具有更高的操作优先级，即应该先访问右操作数进行赋值操作然后再去声明变量，最后进行赋值操作。