Stage-5 Report

曹伦郗 *2020011020*

实验内容

step11

词法分析

在*lex.py*中新增了左右方括号，用于数组声明和索引类。

```
t_LSquareBracket = '['
t_RSquareBracket = ']'
```

语法分析

在*tree.py*中：

- 新增了索引表达式类*IndexExpr*，成员有基址*base*和索引值*index*；
- 完善了声明类*Declaration*，新增成员数组各维大小*indexes*。

```
class IndexExpr(Expression):
    """
    AST node of index expression.
    """

    def __init__(self, base: Union[Identifier, IndexExpr], index: Expression) ->
None:
        super().__init__('index expr')
        self.base = base
        self.index = index

    def __getitem__(self, key: int) -> Node:
        return (self.base, self.index)[key]

    def __len__(self) -> int:
        return 2

    def accept(self, v: Visitor[T, U], ctx: T):
        return v.visitIndexExpr(self, ctx)
```

```
class Declaration(Node):
    """
    AST node of declaration.
    """

    def __init__(
        self,
        var_t: TypeLiteral,
        ident: Identifier,
        indexes: list[IntLiteral],
```

```
        init_expr: Optional[Expression] = None,
    ) -> None:
        super().__init__("declaration")
        self.var_t = var_t
        self.ident = ident
        self.indexes = indexes or NULL
        self.init_expr = init_expr or NULL

    def __getitem__(self, key: int) -> Node:
        return (self.var_t, self.ident, self.indexes, self.init_expr)[key]

    def __len__(self) -> int:
        return 4

    def accept(self, v: Visitor[T, U], ctx: T):
        return v.visitDeclaration(self, ctx)
```

在*Visitor*类下添加了*visitVisitor*。

```
def visitIndexExpr(self, that: IndexExpr, ctx: T) -> Optional[U]:
        return self.visitOther(that, ctx)
```

在*ply_parser.py*中：

- 新增了*p_indexes*、*p_index_empty*，以产生不同个数的索引，产生式为

  ```
  indexes : indexes LSquareBracket Integer RSquareBracket | empty
  ```

  ```
  def p_indexes(p):
      """
      indexes : indexes LSquareBracket Integer RSquareBracket
      """
      p[1].append(p[3])
      p[0] = p[1]


  def p_index_empty(p):
      """
      indexes : empty
      """
      p[0] = []
  ```

- 新增了*p_indexExpr*，产生式为

  ```
  postfix : postfix LSquareBracket expression RSquareBracket
  ```

  ```
  def p_indexExpr(p):
      """
      postfix : postfix LSquareBracket expression RSquareBracket
      """
      p[0] = IndexExpr(p[1], p[3])
  ```

- 修改了*p_declaration*及*p_declaration_init*，以实现声明数组，产生式为

```
declaration : type Identifier indexes | type Identifier indexes Assign
expression
```

```
def p_declaration(p):
    """
    declaration : type Identifier indexes
    """
    p[0] = Declaration(p[1], p[2], p[3])


def p_declaration_init(p):
    """
    declaration : type Identifier indexes Assign expression
    """
    p[0] = Declaration(p[1], p[2], p[3], p[5])
```

- 修改了*p_binary_expression*，以实现对数组元素的赋值。

```
def p_binary_expression(p):
    """
    assignment : unary Assign expression
```

语义分析

修改了*Namer.visitDeclaration*。

根据*decl*的成员数组各维大小*indexes*是否为空判断其是否为数组声明。若不是数组声明，行为与原来一致；否则，创建的变量符号的类型为*ArrayType*，并检查各维大小是否为正数，若不是则抛出*DecafBadArraySizeError*。

```
    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.
        """
        isGlobal = ctx.isGlobalScope()
        if ctx.findConflict(decl.ident.value):
            if isGlobal:
                raise DecafGlobalVarDefinedTwiceError(decl.ident.value)
            raise DecafDeclConflictError(decl.ident.value)
        type = decl.var_t.type if decl.indexes == NULL else ArrayType.multidim(
            decl.var_t.type, *[index.value for index in decl.indexes])
        symbol = VarSymbol(decl.ident.value, type, isGlobal)
        ctx.declare(symbol)
        decl.setattr('symbol', symbol)
        if decl.indexes != NULL:
            for index in decl.indexes:
                if index.value <= 0:
```

```
                    raise DecafBadArraySizeError()
        if decl.init_expr != NULL:
            decl.init_expr.accept(self, ctx)
            if isGlobal and not isinstance(decl.init_expr, IntLiteral):
                raise DecafGlobalVarBadInitValueError(decl.ident.value)
```

新增了*Namer.visitIndexExpr*，即：

①迭代直到找到索引表达式的数组变量符号；

②若变量符号不存在，抛出*DecafUndefinedVarError*；

③若符号不是数组变量符号，抛出*DecafBadIndexError*；

④若索引维数 > 数组维数，抛出*DecafBadIndexError*；

⑤若索引维数 < 数组维数，抛出*DecafTypeMismatchError*（当该方法因*Namer.visitAssignment*被调用时，后者会捕获该异常，转而抛出*DecafBadAssignTypeError*）；

⑥将该变量符号同*indexExpr*关联。

```
    def visitIndexExpr(self, indexExpr: IndexExpr, ctx: ScopeStack) -> None:
        expr = indexExpr
        indexExprDim = 1
        expr.index.accept(self, ctx)
        while not isinstance(expr.base, Identifier):
            expr = expr.base
            indexExprDim += 1
            expr.index.accept(self, ctx)
        arraySymbol = ctx.lookup(expr.base.value)
        if not arraySymbol:
            raise DecafUndefinedVarError(expr.base.value)
        if not isinstance(arraySymbol, VarSymbol) or not isinstance(
            arraySymbol.type, ArrayType):
            raise DecafBadIndexError(expr.base.value)
        if indexExprDim > arraySymbol.type.dim:
            raise DecafBadIndexError(expr.base.value)
        elif indexExprDim < arraySymbol.type.dim:
            raise DecafTypeMismatchError()
        expr.base.setattr('symbol', arraySymbol)
```

修改了*Namer.visitAssignment*。

当左值不是标识符且捕获到*DecafTypeMismatchError*，说明对数组赋值了，故抛出*DecafBadAssignTypeError*。

```
    def visitAssignment(self, expr: Assignment, ctx: ScopeStack) -> None:
        """
        1. Refer to the implementation of visitBinary.
        """
        if not isinstance(expr.lhs, Identifier):
            try:
                expr.lhs.accept(self, ctx)
            except DecafTypeMismatchError:
                raise DecafBadAssignTypeError()
        else:
            expr.lhs.accept(self, ctx)
        expr.rhs.accept(self, ctx)
```

修改了*Namer.visitIdentifier*。

若查找到的变量符号的类型为*ArrayType*，则抛出*DecafBadAssignTypeError*。

```
    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.
        """
        varSymbol = ctx.lookup(ident.value)
        if not varSymbol or not isinstance(varSymbol, VarSymbol):
            raise DecafUndefinedVarError(ident.value)
        if isinstance(varSymbol.type, ArrayType):
            raise DecafBadAssignTypeError()
        ident.setattr('symbol', varSymbol)
```

中间代码生成

新增了*Alloc*指令类，用于为局部数组分配内存空间。

```
class Alloc(TACInstr):
    def __init__(self, dst: Temp, size: int) -> None:
        super().__init__(InstrKind.SEQ, [dst], [], None)
        self.dst = dst
        self.size = size

    def __str__(self) -> str:
        return '%s = alloc %s' % (self.dst, self.size)

    def accept(self, v: TACVisitor) -> None:
        return v.visitAlloc(self)
```

在*tacvisitor.py*中添加了相应函数。

```
   def visitAlloc(self, instr: Alloc) -> None:
        self.visitOther(instr)
```

在*FuncVisitor*中添加相应函数，需要为数组起始地址分配新的临时变量。

```
    def visitAlloc(self, size: int) -> Temp:
        temp = self.freshTemp()
        self.func.add(Alloc(temp, size))
        return temp
```

修改了*TACGen.transform*。

为全局数组设置初始值为全零列表，且列表长度即数组中元素总数，用于在后端添加目标代码中的*data*数据段时区分全局变量和全局数组，并为全局数组分配相应大小内存空间。

```
            if decl.init_expr != NULL:
                globalSymbol.setInitValue(decl.init_expr.value)
            elif isinstance(globalSymbol.type, ArrayType): # default zero
initialization, both variable and array
                globalSymbol.initValue = [globalSymbol.initValue] *
int(globalSymbol.type.size / 4)
            globalSymbolNameValues[globalSymbol.name] = globalSymbol.initValue
```

修改了*TACGen.visitDeclaration*。

对数组声明，数组变量符号的临时变量为*mv.visitAlloc*的返回值。此外，为*TACFunc*新增成员*arrays*，记录局部数组的起始地址和大小。

```
        if decl.indexes == NULL:
            symbol.temp = mv.freshTemp()
        else:
            symbol.temp = mv.visitAlloc(symbol.type.size)
            mv.func.arrays.append((symbol.temp, symbol.type.size))
```

新增了*TACGen.addressCompute*，即：

①迭代直到找到索引表达式的数组变量符号，过程中保存索引表达式的各维索引值；

②若该数组为全局数组，调用*mv.visitLoadSymbol*以获得其起始地址；

②迭代数组变量符号的类型，直到不再是*ArrayType*，过程中保存数组各维大小，最低维大小为*1*；

③从低维到高维，通过累乘获得每一维的元素大小，并累加各维索引与该维的元素大小的乘积作为偏移量，与数组起始地址求和，返回索引表达式对应数组元素的地址。

```
    def addressCompute(self, indexExpr: IndexExpr, mv: FuncVisitor) -> Temp:
        expr = indexExpr
        indexes = []
        expr.index.accept(self, mv)
        indexes.append(expr.index.getattr('val'))
        while not isinstance(expr.base, Identifier):
            expr = expr.base
            expr.index.accept(self, mv)
            indexes.append(expr.index.getattr('val'))
        arraySymbol = expr.base.getattr('symbol')
        if arraySymbol.isGlobal:
            arraySymbol.temp = mv.visitLoadSymbol(arraySymbol.name)
        lengths = []
        type = arraySymbol.type
        while isinstance(type, ArrayType):
```

```
            lengths.append(type.length)
            type = type.base
        lengths.append(1)
        lengths.reverse()
        addrTemp = arraySymbol.temp
        size = 4
        for i, index in enumerate(indexes):
            size *= lengths[i]
            addrTemp = mv.visitBinary(tacop.BinaryOp.ADD, addrTemp,
mv.visitBinary(
                tacop.BinaryOp.MUL, index, mv.visitLoad(size)
            ))
        return addrTemp
```

新增了*TACGen.visitIndexExpr*，即：

①通过*TACGen.addressCompute*获得索引表达式对应数组元素的地址；

②调用*mv.visitLoadInMem*以获取该数组元素，并将其作为该索引表达式的值。

```
    def visitIndexExpr(self, indexExpr: IndexExpr, mv: FuncVisitor) -> None:
        addrTemp = self.addressCompute(indexExpr, mv)
        indexExpr.setattr('val', (mv.visitLoadInMem(addrTemp, 0)))
```

修改了*TACGen.visitAssignment*。

对于左值是索引表达式的情况，通过*TACGen.addressCompute*获得索引表达式对应数组元素的地址后，调用*mv.visitStoreInMem*完成对数组元素的赋值。

```
    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.
        """
        expr.rhs.accept(self, mv)
        if isinstance(expr.lhs, IndexExpr):
            addrTemp = self.addressCompute(expr.lhs, mv)
            mv.visitStoreInMem(expr.rhs.getattr('val'), addrTemp, 0)
            expr.setattr('val', expr.rhs.getattr('val'))
        else:
            symbol = expr.lhs.getattr('symbol')
            if symbol.isGlobal:
                base = mv.visitLoadSymbol(symbol.name)
                mv.visitStoreInMem(expr.rhs.getattr('val'), base, 0)
                expr.setattr('val', expr.rhs.getattr('val'))
            else:
                expr.setattr('val', mv.visitAssignment(symbol.temp,
expr.rhs.getattr('val')))
```

目标平台汇编代码生成

新增了*Riscv.LoadArray*指令。

它使目的寄存器自增*SP*的值，当为它生成的*Loc*分配寄存器时，还会在它之前添加*Riscv.LoadImm*汇编指令，以将该目的寄存器设置预设为数组起始地址相对*SP*的偏移量，从而和*SP*求和后，目的寄存器中即为数组起始地址。

```
    class LoadArray(TACInstr):
        def __init__(self, dst: Temp) -> None:
            super().__init__(InstrKind.SEQ, [dst], [dst, Riscv.SP], None)
            self.dst = dst

        def __str__(self) -> str:
            return "add " + Riscv.FMT3.format(
                str(self.dsts[0]), str(self.srcs[0]), str(self.srcs[1])
            )
```

在*RiscvInstrSelector*下实现了*visitAlloc*。

```
        # in step11, you need to think about how to store the array
        def visitAlloc(self, instr: Alloc) -> None:
            self.seq.append(Riscv.LoadArray(instr.dst))
```

在*RiscvAsmEmitter*的初始化中，增加对全局数组的声明。

```
                if isinstance(initValue, int): # global variable
                    self.printer.println('.word %s' % (initValue))
                else: # global array
                    self.printer.println('.zero %s' % (len(initValue) * 4))
```

在*RiscvSubroutineEmitter*的初始化中，添加了成员*arrays*，其来源为对应的*TACFunc*的成员*arrays*，保存了该函数的每个局部数组的起始地址及大小，并为它们计算出其同*SP*的偏移量，保存在成员*arraySPOffsets*中。

```
        # offset to SP of each local array of this function
        self.arraySPOffsets = {}
        for addrTemp, size in arrays:
            self.arraySPOffsets[addrTemp.index] = self.nextLocalOffset
            self.nextLocalOffset += size
```

寄存器分配

新增了*allocForCall*，用于在访问到由*Riscv.LoadArray*生成的*Loc*时取代*allocForLoc*。

该方法内部逻辑如下：

①添加*Riscv.LoadImm*汇编指令，以将该目的寄存器设置预设为数组起始地址相对*SP*的偏移量；

②为该条由*Riscv.LoadArray*生成的*Loc*分配寄存器，使目的寄存器和*SP*求和，目的寄存器中即为数组起始地址。

```
        # in step9, you may need to think about how to store callersave regs
here
        for loc in bb.allSeq():
            subEmitter.emitComment(str(loc.instr))
            if isinstance(loc.instr, Riscv.Call):
                self.allocForCall(loc, subEmitter)
            elif isinstance(loc.instr, Riscv.LoadArray):
                self.allocForLoadArray(loc, subEmitter)
            else:
                self.allocForLoc(loc, subEmitter)
```

```
    def allocForLoadArray(self, loc: Loc, subEmitter: SubroutineEmitter):
        reg = self.allocRegFor(loc.instr.dst, False, loc.liveIn, subEmitter)
        subEmitter.emitNative(Riscv.LoadImm(reg,
subEmitter.arraySPOffsets[loc.instr.dst.index]))
        self.allocForLoc(loc, subEmitter)
```

step12

语法分析

在*tree.py*中：

- 完善了声明类*Declaration*，新增成员初始化列表*init_list*。
- 完善了形参类*Parameter*，新增成员*isArray*、*indexes*，以实现数组类形参，且最高维大小被忽略。

```
class Declaration(Node):
    """
    AST node of declaration.
    """

    def __init__(
        self,
        var_t: TypeLiteral,
        ident: Identifier,
        indexes: list[IntLiteral],
        init_expr: Optional[Expression] = None,
        init_list: Optional[list[IntLiteral]] = None
    ) -> None:
        super().__init__("declaration")
        self.var_t = var_t
        self.ident = ident
        self.indexes = indexes or NULL
        self.init_expr = init_expr or NULL
        self.init_list = init_list or NULL

    def __getitem__(self, key: int) -> Node:
        return (self.var_t, self.ident, self.indexes, self.init_expr,
self.init_list)[key]

    def __len__(self) -> int:
        return 5

    def accept(self, v: Visitor[T, U], ctx: T):
        return v.visitDeclaration(self, ctx)
```

```
class Parameter(Node):
    """
    AST node that represents a parameter.
    """

    def __init__(self, var_t: TypeLiteral, ident: Identifier, isArray: bool,
indexes: list[IntLiteral]) -> None:
        super().__init__('parameter')
        self.var_t = var_t
        self.ident = ident
        self.isArray = isArray
        self.indexes = indexes or NULL

    def __getitem__(self, key: int) -> Node:
        return (self.var_t, self.ident, self.isArray, self.indexes)[key]

    def __len__(self) -> int:
        return 4

    def accept(self, v: Visitor[T, U], ctx: T):
        return v.visitParameter(self, ctx)
```

在*ply_parser.py*中：

- 修改了*p_parameter*，新增了*p_parameter_array_empty*、*p_parameter_array_not_empty*，以实现数组类形参，且最高维大小被忽略，产生式为

  ```
  parameter : type Identifier | type Identifier LSquareBracket RSquareBracket
  indexes | type Identifier LSquareBracket Integer RSquareBracket indexes
  ```

  ```
  def p_parameter(p):
      """
      parameter : type Identifier
      """
      p[0] = Parameter(p[1], p[2], False, [])


  def p_parameter_array_empty(p):
      """
      parameter : type Identifier LSquareBracket RSquareBracket indexes
      """
      p[0] = Parameter(p[1], p[2], True, p[5])


  def p_parameter_array_nonempty(p):
      """
      parameter : type Identifier LSquareBracket Integer RSquareBracket
  indexes
      """
      p[0] = Parameter(p[1], p[2], True, p[6])
  ```

- 新增了*p_integer_list*、*p_integer_list_empty*、*p_integer_list_nonempty*、*p_integer_list_one*，以产生不同元素个数的初始化列表，产生式为

```
integer_list : integer_list_nonempty | empty
integer_list_nonempty : integer_list_nonempty Comma Integer | Integer
```

```
def p_integer_list(p):
    """
    integer_list : integer_list_nonempty
    """
    p[0] = p[1]


def p_integer_list_empty(p):
    """
    integer_list : empty
    """
    p[0] = []


def p_integer_list_nonempty(p):
    """
    integer_list_nonempty : integer_list_nonempty Comma Integer
    """
    p[1].append(p[3])
    p[0] = p[1]


def p_integer_list_one(p):
    """
    integer_list_nonempty : Integer
    """
    p[0] = [p[1]]
```

- 将*p_declaration_init*改为*p_declaration_var*作为带有初始值的变量声明，新增了*p_declaration_array*作为带有初始化列表的数组声明，产生式为

```
declaration : type Identifier Assign expression | type Identifier indexes
Assign LBrace integer_list RBrace
```

```
def p_declaration_init_var(p):
    """
    declaration : type Identifier Assign expression
    """
    p[0] = Declaration(p[1], p[2], [], p[4], [])
```

```
def p_declaration_init_array(p):
    """
    declaration : type Identifier indexes Assign LBrace integer_list RBrace
    """
    p[0] = Declaration(p[1], p[2], p[3], NULL, p[6])
```

语义分析

修改了*Namer.visitFunction*。

在为函数符号添加形参类型时，须根据形参的成员*isArray*判断是否为数组，从而生成正确的形参类型。

```
            else: # not declared, declare and define at the same time
                funcSymbol = FuncSymbol(func.ident.value, func.ret_t.type,
ctx.currentScope(), True)
                for param in func.parameter_list:
                    if not param.isArray:
                        funcSymbol.addParaType(param.var_t.type)
                    else:
                        funcSymbol.addParaType(ArrayType.multidim(
                            param.var_t.type, 1, *[index.value for index in
param.indexes]
                        ))
```

```
            # first declaration
            funcSymbol = FuncSymbol(func.ident.value, func.ret_t.type,
ctx.currentScope(), False)
            for param in func.parameter_list:
                if not param.isArray:
                    funcSymbol.addParaType(param.var_t.type)
                else:
                    funcSymbol.addParaType(ArrayType.multidim(
                        param.var_t.type, 1, *[index.value for index in
param.indexes]
                    ))
```

修改了*Namer.visitParameter*。

根据成员*isArray*判断是否为数组。若形参是数组，创建的变量符号的类型为*ArrayType*，并检查各维大小是否为正数，若不是则抛出*DecafBadArraySizeError*。

```
    def visitParameter(self, param: Parameter, ctx: ScopeStack) -> None:
        if ctx.findConflict(param.ident.value):
            raise DecafDeclConflictError(param.ident.value)
        type = param.var_t.type if not param.isArray else ArrayType.multidim(
            param.var_t.type, 1, *[index.value for index in param.indexes])
        symbol = VarSymbol(param.ident.value, type)
        ctx.declare(symbol)
        param.setattr('symbol', symbol)
        if param.isArray:
            for index in param.indexes:
                if index.value <= 0:
                    raise DecafBadArraySizeError()
```

修改了*Namer.visitCall*。

检查各个参数时，若参数是标识符，则需检查其类型是否与对应形参的类型相同，其中若同为数组，则需检查*ArrayType.base*是否一致。

```
        for i, argument in enumerate(call.argument_list):
            if isinstance(argument, Identifier):
                varSymbol = ctx.lookup(argument.value)
                if not varSymbol or not isinstance(varSymbol, VarSymbol):
                    raise DecafUndefinedVarError(argument.value)
                if isinstance(funcSymbol.getParaType(i), ArrayType) or
isinstance(varSymbol.type, ArrayType):
                    if not isinstance(funcSymbol.getParaType(i), ArrayType) or
not isinstance(
                            varSymbol.type, ArrayType) or
funcSymbol.getParaType(i).indexed != varSymbol.type.indexed:
                        raise DecafTypeMismatchError()
                argument.setattr('symbol', varSymbol)
            else:
                argument.accept(self, ctx)
```

修改了*Namer.visitDeclaration*。

对数组进行初始化。若有初始化列表，检查是否超过声明的大小，若合法，则将初始化列表作零扩展后，设为数组变量符号的初始值，否则抛出*DecafTypeMismatchError*；若没有初始化列表，则进行零初始化，以一个元组作为数组变量符号的初始值，元组中保存一个*0*和数组元素总数。通过判断数组变量符号的初始值是列表还是元组，即可区分是带有初始化列表还是零初始化。

```
        if decl.indexes != NULL:
            for index in decl.indexes:
                if index.value <= 0:
                    raise DecafBadArraySizeError()
            if decl.init_list != NULL:
                if len(decl.init_list) * 4 > type.size:
                    raise DecafTypeMismatchError()
                symbol.initValue = [integer.value for integer in decl.init_list]
                symbol.initValue.extend([0] * int(type.size / 4 -
len(decl.init_list)))
            else:
                # 0 means ZERO initialization, which is used in
TACGen.visitDeclaration, check 'if 0 in symbol.initValue'
                symbol.initValue = (0, int(type.size / 4))
        elif decl.init_expr != NULL:
            decl.init_expr.accept(self, ctx)
            if isGlobal and not isinstance(decl.init_expr, IntLiteral):
                raise DecafGlobalVarBadInitValueError(decl.ident.value)
```

中间代码生成

删除了原本*TACGen.transform*中对全局数组的变量符号设初始值的行为，该行为在上一步中由*Namer.visitDeclaration*执行。

修改了*TACGen.visitIdentifier*。

由于该方法会在传递全局数组作参数时，因*TACGen.visitCall*被调用，需要调用*mv.visitLoadSymbol*，提供全局数组的变量符号的临时变量。

```
    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.
        """
        symbol = ident.getattr('symbol')
        if symbol.isGlobal:
            base = mv.visitLoadSymbol(symbol.name)
            if not isinstance(symbol.type, ArrayType):
                symbol.temp = mv.visitLoadInMem(base, 0)
            else:
                symbol.temp = base
        ident.setattr('val', symbol.temp)
```

修改了*TACGen.visitDeclaration*。

根据上一步中*Namer.visitDeclaration*为数组变量符号设置的初始值，为局部数组声明添加初始化。考虑到初始化列表可能进行了零扩展，或数组自动零初始化，故如果发现数组的初始值中有*0*，则提前准备一个临时变量保存*0*，避免在循环中重复调用*mv.visitLoad(0)*。

```
        if decl.indexes == NULL:
            symbol.temp = mv.freshTemp()
        else:
            symbol.temp = mv.visitAlloc(symbol.type.size)
            offset = 0
            if 0 in symbol.initValue:
                zeroTemp = mv.visitLoad(0)
            for integer in symbol.initValue:
                temp = mv.visitLoad(integer) if integer != 0 else zeroTemp
                mv.visitStoreInMem(temp, symbol.temp, offset)
                offset += 4
            mv.func.arrays.append((symbol.temp, symbol.type.size))
```

目标平台汇编代码生成

在*RiscvAsmEmitter*的初始化中，修改对全局数组的声明，对有初始化列表和进行零初始化的全局数组作不同处理。

```
                else: # global array
                    if isinstance(initValue, list):
                        for integer in initValue:
                            self.printer.println('.word %s' % (integer))
                    else:
                        self.printer.println('.zero %s' % (initValue[1] * 4))
```

思考题

step11

不能再在函数起始阶段开辟栈帧时，就为局部数组分配好栈上的内存空间。应变为

- 在中间代码生成阶段，对每条变长的一维局部数组声明，添加一条新设计的*TAC*指令，它的目的临时变量保存了该数组的起始地址，源临时变量保存了该变长数组的大小，形如

```
dstTemp = VLA_Alloc srcTemp
```

其被后端翻译为汇编代码如

```
li t0, -4
mul t1, t1, t0
add sp, sp, t1
mv t2, sp
```

其中，*t2*为该*TAC*指令*dstTemp*的寄存器，用于保存该数组的起始地址；*t1*为该*TAC*指令*srcTemp*的寄存器，用于保存该数组的大小，此处仅为一个示例。

- 而函数在设置好返回值之后，在恢复*calleesave*寄存器之前，需要借助*FP*来恢复*SP*。

  具体汇编指令为

  ```
  addi sp, fp, -64
  ```

  其中，*64*为函数创建新栈帧时，*SP*减少的大小，此处仅为一个示例。

step12

因为索引表达式在计算数组元素的地址时，总是用该维度上的索引值×更低的所有维度的大小的乘积（最低维索引直接对应单个元素，故大小为*1*），因此不会用到最高维（第一维）的大小。

例如*a[3][4][5]*这个数组被作为函数参数，当访问*a[x][y][z]*时，地址计算为

$$a + [z + y \times 5 + x \times (4 \times 5)]$$

其中并不涉及对数组*a*最高维（第一维）大小*3*的使用，故作为函数参数的数组类型第一维的大小无关紧要，可以为空。