Stage-3 : 作用域和循环

陈炫中　2019011236

- **Step 7**

实验内容

增加块语句的支持。

在 `frontend/typecheck/namer.py` 的 `visitBlock` 函数中，因为可能存在多个块语句，因此对原来的写法进行如下修改：在构建符号表阶段访问块语句时，通过 `Scope(ScopeKind.LOCAL)` 为每个语句块分别开启一个局部作用域，并且在访问结束后，关闭此作用域。具体写法如下：

```
    def visitBlock(self, block: Block, ctx: ScopeStack) -> None:
        for child in block:
            if child.name == "block":
                scope = Scope(ScopeKind.LOCAL)
                ctx.open(scope)
                child.accept(self, ctx)
                ctx.close()
            else:
                child.accept(self, ctx)
```

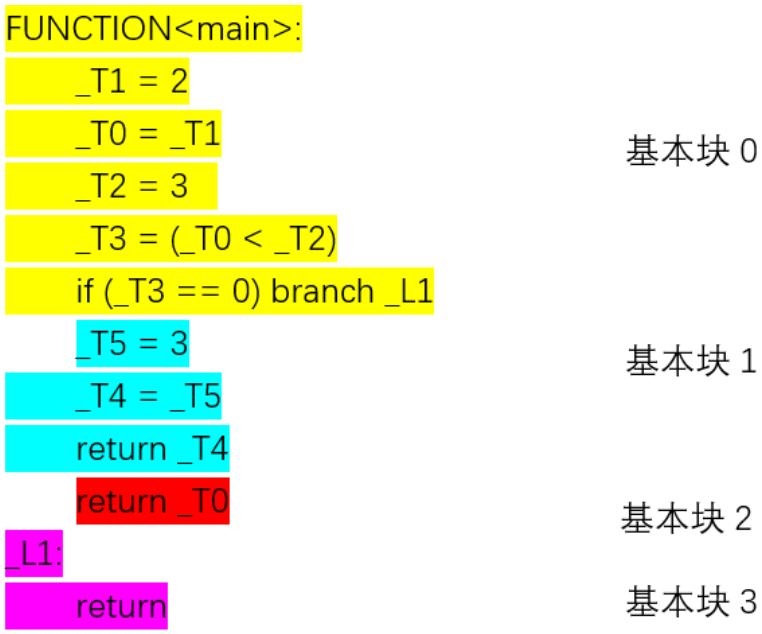
引入块语句之后，寄存器分配部分同样需要作出修改：在 `backend/reg/bruteregalloc.py` 中，不再为不可达的基本块分配寄存器，我的做法是对当前基本块在 CFG 中的入度进行判断，如果其入度为 0 且其不为 0 号基本块，则当前基本块一定是不可达的，将其跳过即可，具体写法如下：

```
    def accept(self, graph: CFG, info: SubroutineInfo) -> None:
        subEmitter = self.emitter.emitSubroutine(info)
        for bb in graph.iterator():
            # you need to think more here
            # maybe we don't need to alloc regs for all the basic blocks
            if bb.label is not None:
                subEmitter.emitLabel(bb.label)
            reachable = True
            if graph.getInDegree(bb.id) == 0 and bb.id != 0:
                reachable = False
            if reachable:
                self.localAlloc(bb, subEmitter)
        subEmitter.emitEnd()
```

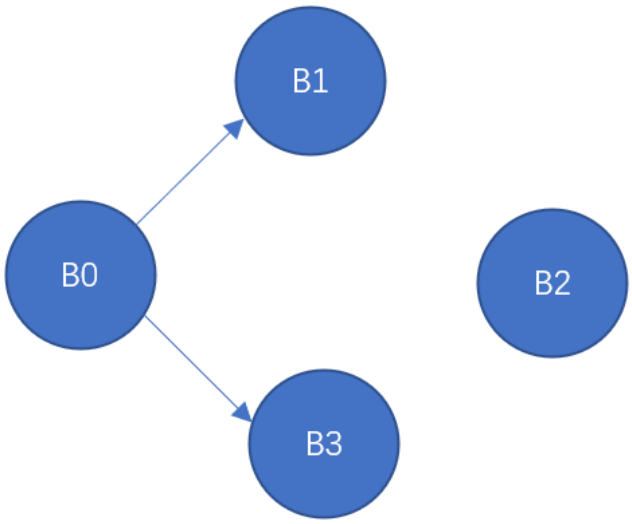
思考题

```
int main(){
 int a = 2;
 if (a < 3) {
     {
         int a = 3;
         return a;
     }
     return a;
 }
}
```

首先划分基本块：



则上述 MiniDecaf 代码的控制流图如下：



- **Step 8**

实验内容

增加对循环语句，以及 break/continue 的支持。

仿照 While 的实现，在 `frontend/ast/tree.py` 中，新增 For 和 DoWhile 节点，以 For 为例，具体实现如下：

```
class For(Statement):
    """
    AST node of for statement.
    """

    def __init__(self, init: Expression, cond: Expression, update: Expression,
body: Statement) -> None:
        super().__init__("for")
        self.init = init
        self.cond = cond
        self.update = update
        self.body = body

    def __getitem__(self, key: int) -> Node:
        return (self.init, self.cond, self.update, self.body)[key]

    def __len__(self) -> int:
        return 4

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

Continue 节点的增加同理仿照 Break 进行实现，不再赘述。

以 For 为例，在 `frontend/ast/visitor.py` 中新增的节点 Visitor 的默认函数：

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

在词法上，在 `frontend/lex/lex.py` 中增加保留字：

```
    "for": "For",
    "continue": "Continue",
    "do": "Do",
```

在语法上，在 `frontend/parser/ply_parser.py` 中，仿照 `p_while` 的实现，添加对应的 `p_for/p_dowhile/p_continue` 函数，以 `p_for` 为例，具体实现如下：

```
def p_for(p):
    """
    statement : For LParen block_item block_item opt_expression RParen statement
    """
    p[0] = For(p[3], p[4], p[5], p[7])
```

在中间代码生成上，在 `frontend/tacgen/tacgen.py` 中仿照 `visitWhile` 的实现，增加对应的 `visitFor/visitDoWhile/visitContinue` 函数，以 `visitFor` 为例，具体实现如下：

```
    def visitFor(self, stmt: For, mv: FuncVisitor) -> None:
        #TODO
        beginLabel = mv.freshLabel()
        loopLabel = mv.freshLabel()
        breakLabel = mv.freshLabel()
        mv.openLoop(breakLabel, loopLabel)
```

```
        if stmt.init is not NULL:
            stmt.init.accept(self, mv)
        mv.visitLabel(beginLabel)
        stmt.cond.accept(self, mv)
        if stmt.cond is not NULL:
            mv.visitCondBranch(tacop.CondBranchOp.BEQ, stmt.cond.getattr("val"),
breakLabel)

        stmt.body.accept(self, mv)
        mv.visitLabel(loopLabel)
        if stmt.update is not NULL:
            stmt.update.accept(self, mv)
        mv.visitBranch(beginLabel)
        mv.visitLabel(breakLabel)
        mv.closeLoop()
```

思考题

1.第二种翻译方式更好。第一种翻译方式在退出循环体时需要跳转两次（`br BEGINLOOP_LABEL`，`beqz BREAK_LABEL`），而第二种翻译方式则不需要这两次跳转。
