Kaleidoscope

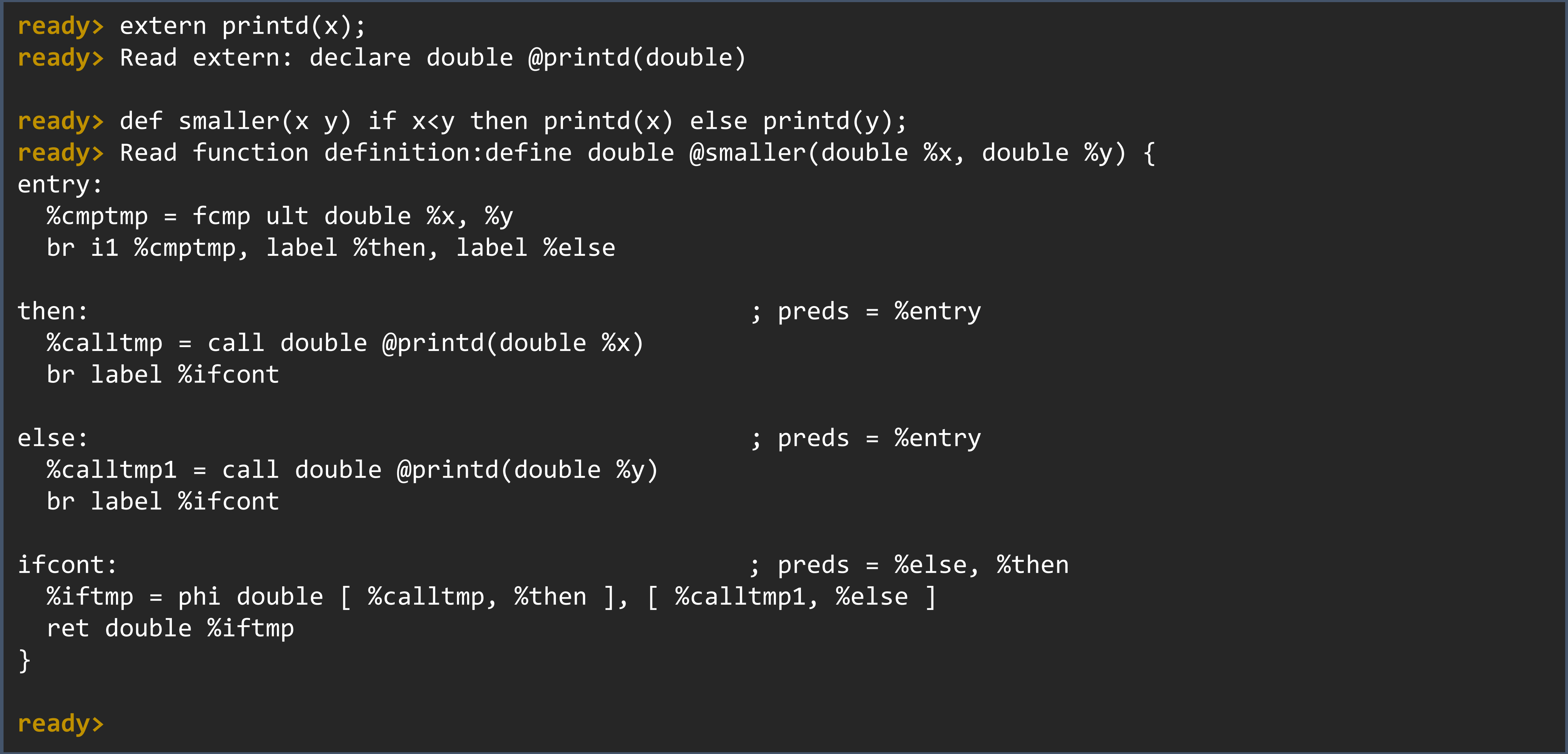
代码解释(4)

万花筒语言 - LLVM 新手入门教程

https://llvm.org/docs/tutorial/MyFirstLanguageFrontend/LangImpl05.html

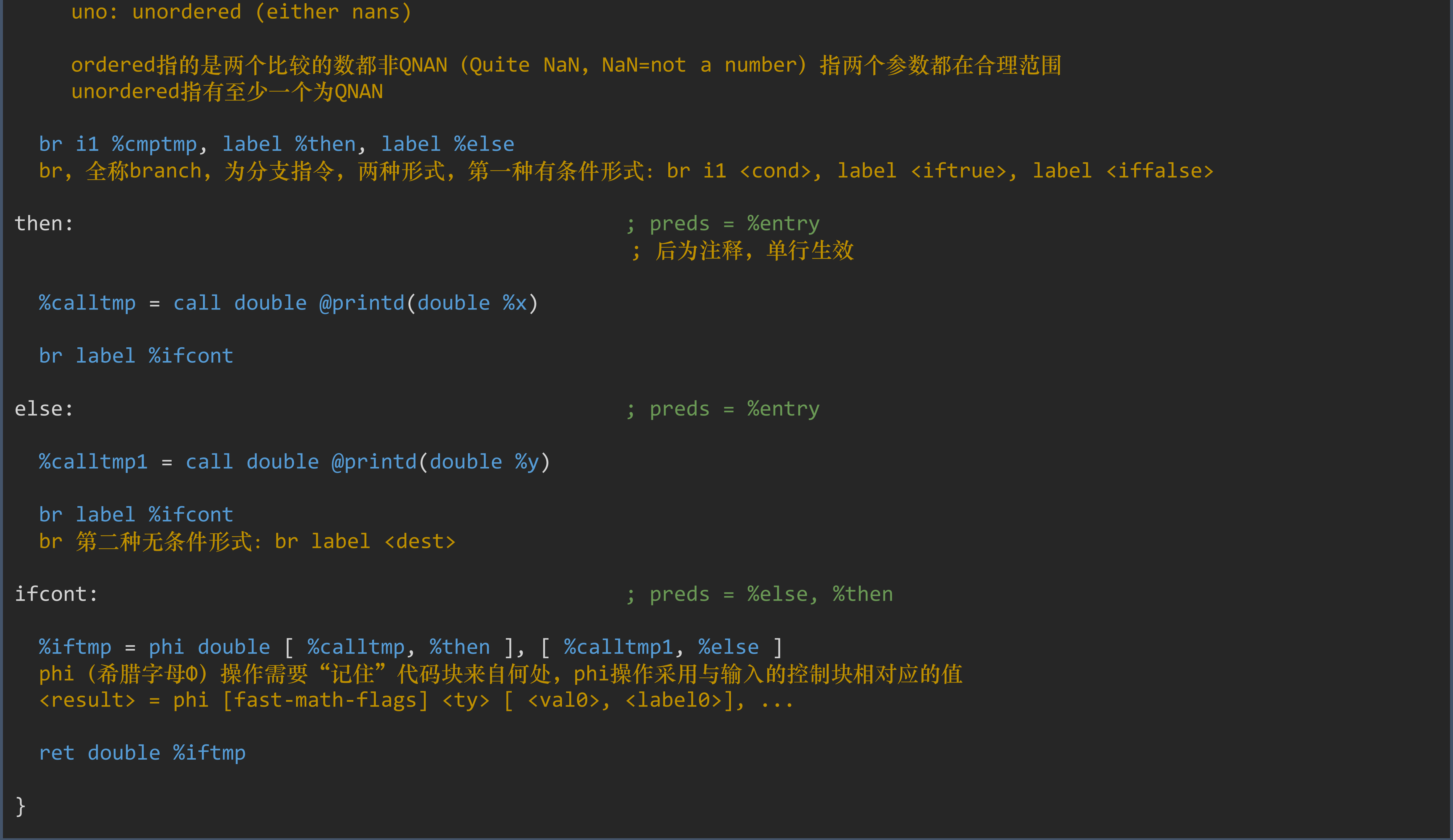
PLCT - SSC

def smaller(x y) if x<y then x else y;



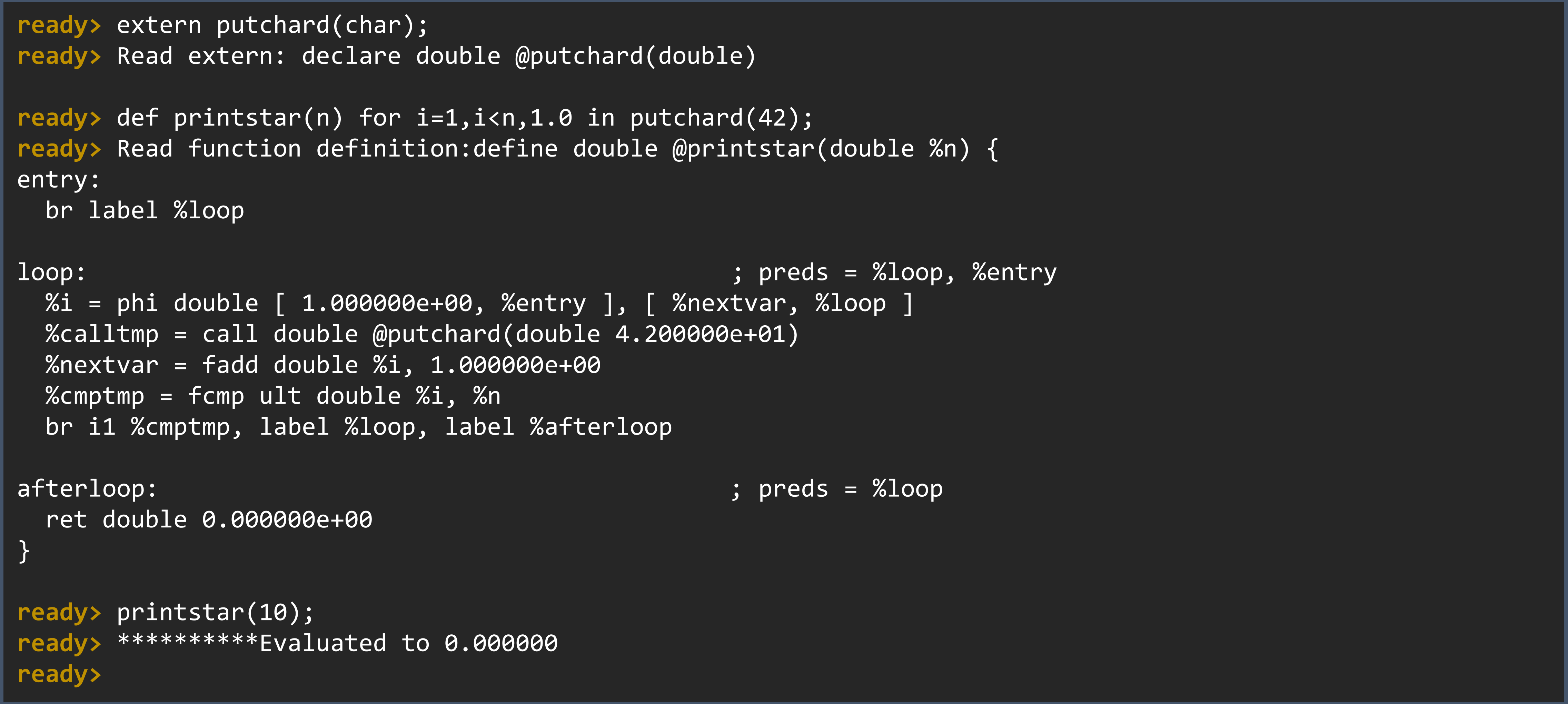
```
ready> extern printd(x);
ready> Read extern: declare double @printd(double)

ready> def smaller(x y) if x<y then printd(x) else printd(y);
ready> Read function definition:define double @smaller(double %x, double %y) {
entry:
  %cmptmp = fcmp ult double %x, %y
  br i1 %cmptmp, label %then, label %else

then:                                             ; preds = %entry
  %calltmp = call double @printd(double %x)
  br label %ifcont

else:                                             ; preds = %entry
  %calltmp1 = call double @printd(double %y)
  br label %ifcont

ifcont:                                           ; preds = %else, %then
  %iftmp = phi double [ %calltmp, %then ], [ %calltmp1, %else ]
  ret double %iftmp
}

ready>
```

```
define double @smaller(double %x, double %y) {
entry:
标签
  %cmptmp = fcmp ult double %x, %y
  fcmp, float compare, 返回一个i1 (Boolean, 布尔) 或者i1的向量 (vector) , 取决于参数类型
  <result> = fcmp [fast-math flags]* <cond> <ty> <op1>, <op2>     ; yields i1 or <N x i1>:result

     false: no comparison, always returns false
     true: no comparison, always returns true

     oeq: ordered and equal
     ogt: ordered and greater than
     oge: ordered and greater than or equal
     olt: ordered and less than
     ole: ordered and less than or equal
     one: ordered and not equal
     ord: ordered (no nans)

     ueq: unordered or equal
     ugt: unordered or greater than
     uge: unordered or greater than or equal
     ult: unordered or less than
     ule: unordered or less than or equal
     une: unordered or not equal
     uno: unordered (either nans)

     ordered指的是两个比较的数都非QNAN (Quite NaN, NaN=not a number) 指两个参数都在合理范围
     unordered指有至少一个为QNAN

  br i1 %cmptmp, label %then, label %else
  br, 全称branch, 为分支指令, 两种形式, 第一种有条件形式: br i1 <cond>, label <iftrue>, label <iffalse>
```



```
    uno: unordered (either nans)

    ordered指的是两个比较的数都非QNAN (Quite NaN, NaN=not a number) 指两个参数都在合理范围
    unordered指有至少一个为QNAN

  br i1 %cmptmp, label %then, label %else
  br, 全称branch, 为分支指令, 两种形式, 第一种有条件形式: br i1 <cond>, label <iftrue>, label <iffalse>

then:                                             ; preds = %entry
                                                  ; 后为注释, 单行生效

  %calltmp = call double @printd(double %x)

  br label %ifcont

else:                                             ; preds = %entry

  %calltmp1 = call double @printd(double %y)

  br label %ifcont
  br 第二种无条件形式: br label <dest>

ifcont:                                           ; preds = %else, %then

  %iftmp = phi double [ %calltmp, %then ], [ %calltmp1, %else ]
  phi (希腊字母Φ) 操作需要“记住”代码块来自何处, phi操作采用与输入的控制块相对应的值
  <result> = phi [fast-math-flags] <ty> [ <val0>, <label0>], ...

  ret double %iftmp

}
```

def printstar(n) for i=1,i<n,1.0 in putchard(42);



```
ready> extern putchard(char);
ready> Read extern: declare double @putchard(double)

ready> def printstar(n) for i=1,i<n,1.0 in putchard(42);
ready> Read function definition:define double @printstar(double %n) {
entry:
  br label %loop

loop:                                             ; preds = %loop, %entry
  %i = phi double [ 1.000000e+00, %entry ], [ %nextvar, %loop ]
  %calltmp = call double @putchard(double 4.200000e+01)
  %nextvar = fadd double %i, 1.000000e+00
  %cmptmp = fcmp ult double %i, %n
  br i1 %cmptmp, label %loop, label %afterloop

afterloop:                                        ; preds = %loop
  ret double 0.000000e+00
}

ready> printstar(10);
ready> **********Evaluated to 0.000000
ready>
```

预处理

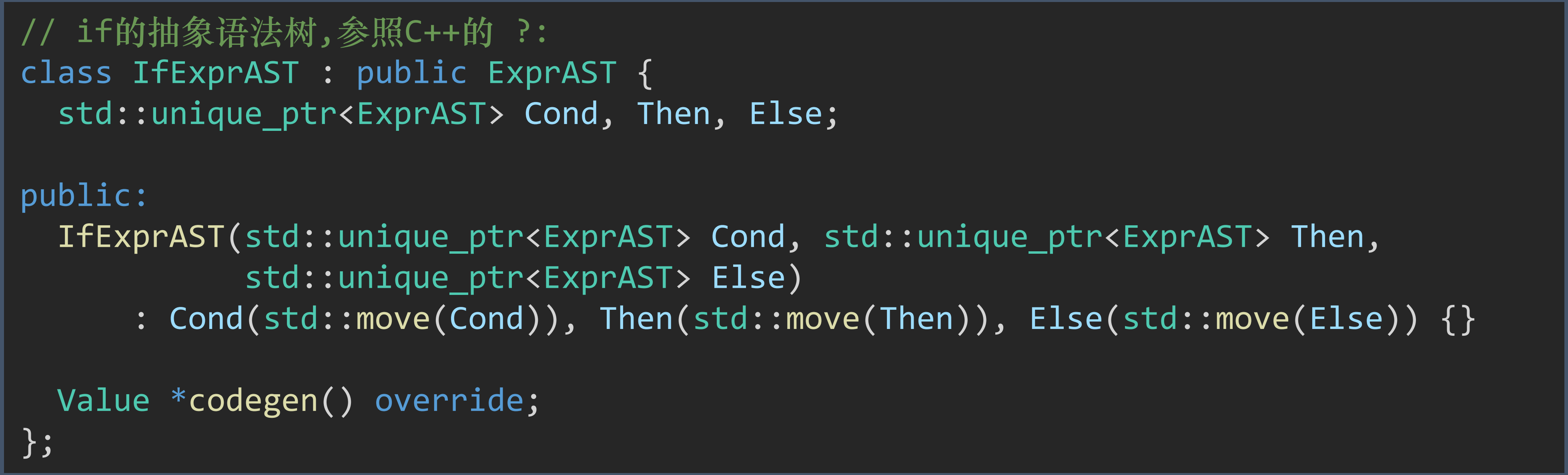
```
#include "llvm/IR/Instructions.h"
// Instructions.h: 定义了 Instruction class 的子类
// Instruction.h: 所有 LLVM 指令的基类
```

词法分析

```
enum Token {

  // control
  // 增加if和for相关的token
  tok_if = -6,
  tok_then = -7,
  tok_else = -8,
  tok_for = -9,
  tok_in = -10
};
```

```
static int gettok() {
  static int LastChar = ' ';

  // Skip any whitespace.
  while (isspace(LastChar))
    LastChar = getchar();


    if (IdentifierStr == "def")
      return tok_def;
    if (IdentifierStr == "extern")
      return tok_extern;
    // 处理相关的 关键字,以返回相应的token
    if (IdentifierStr == "if")
      return tok_if;
    if (IdentifierStr == "then")
      return tok_then;
    if (IdentifierStr == "else")
      return tok_else;
    if (IdentifierStr == "for")
      return tok_for;
    if (IdentifierStr == "in")
      return tok_in;
    return tok_identifier;
  }
}
```

if AST

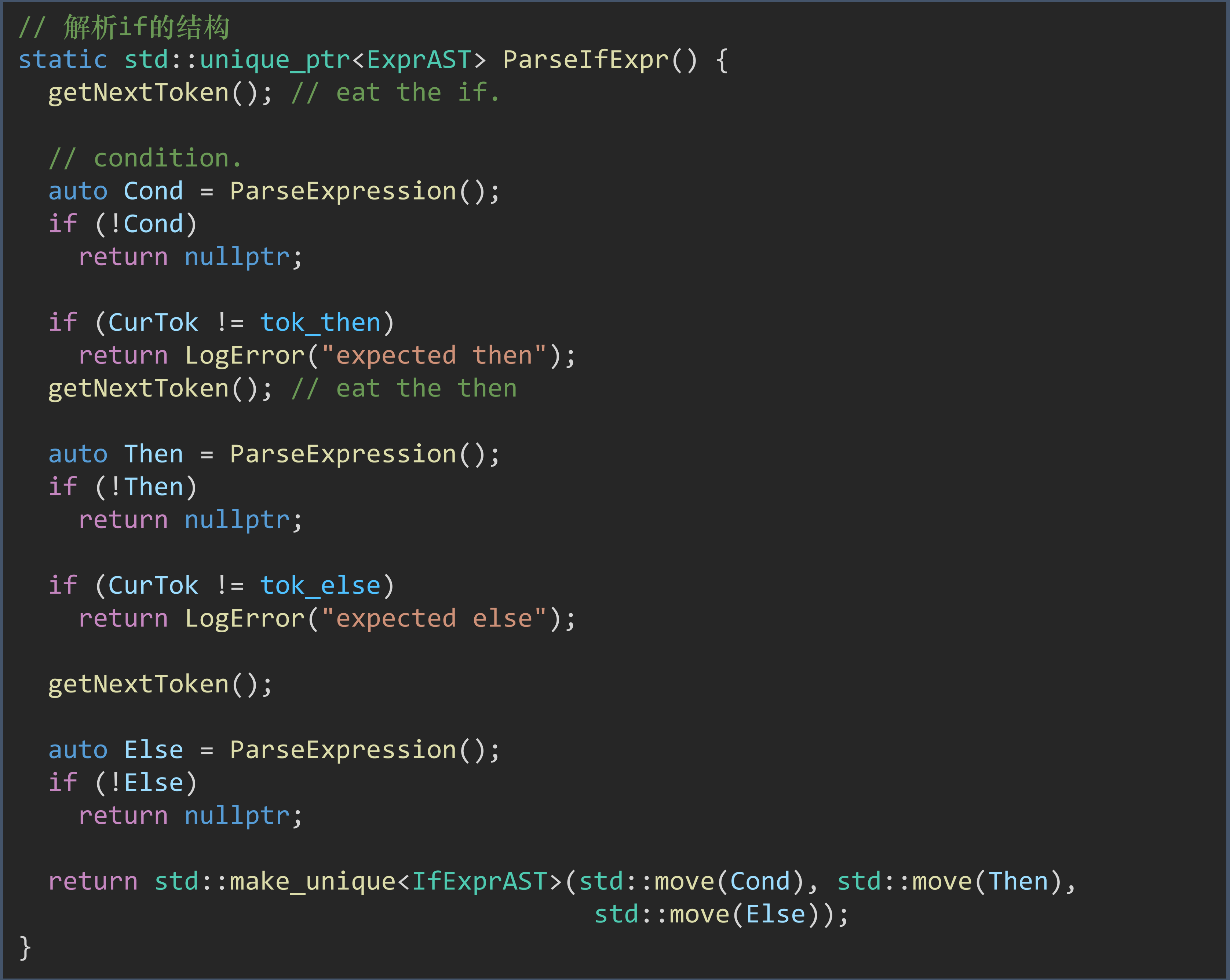


```
// if的抽象语法树,参照C++的 ?:
class IfExprAST : public ExprAST {
  std::unique_ptr<ExprAST> Cond, Then, Else;

public:
  IfExprAST(std::unique_ptr<ExprAST> Cond, std::unique_ptr<ExprAST> Then,
            std::unique_ptr<ExprAST> Else)
      : Cond(std::move(Cond)), Then(std::move(Then)), Else(std::move(Else)) {}

  Value *codegen() override;
};
```

for AST

```
// for的抽象语法树,参照C++的 for(double i=Start;i!=End;i+=Step)
class ForExprAST : public ExprAST {
  std::string VarName;
  std::unique_ptr<ExprAST> Start, End, Step, Body;

public:
  ForExprAST(const std::string &VarName, std::unique_ptr<ExprAST> Start,
             std::unique_ptr<ExprAST> End, std::unique_ptr<ExprAST> Step,
             std::unique_ptr<ExprAST> Body)
      : VarName(VarName), Start(std::move(Start)), End(std::move(End)),
        Step(std::move(Step)), Body(std::move(Body)) {}

  Value *codegen() override;
};
```

if parser

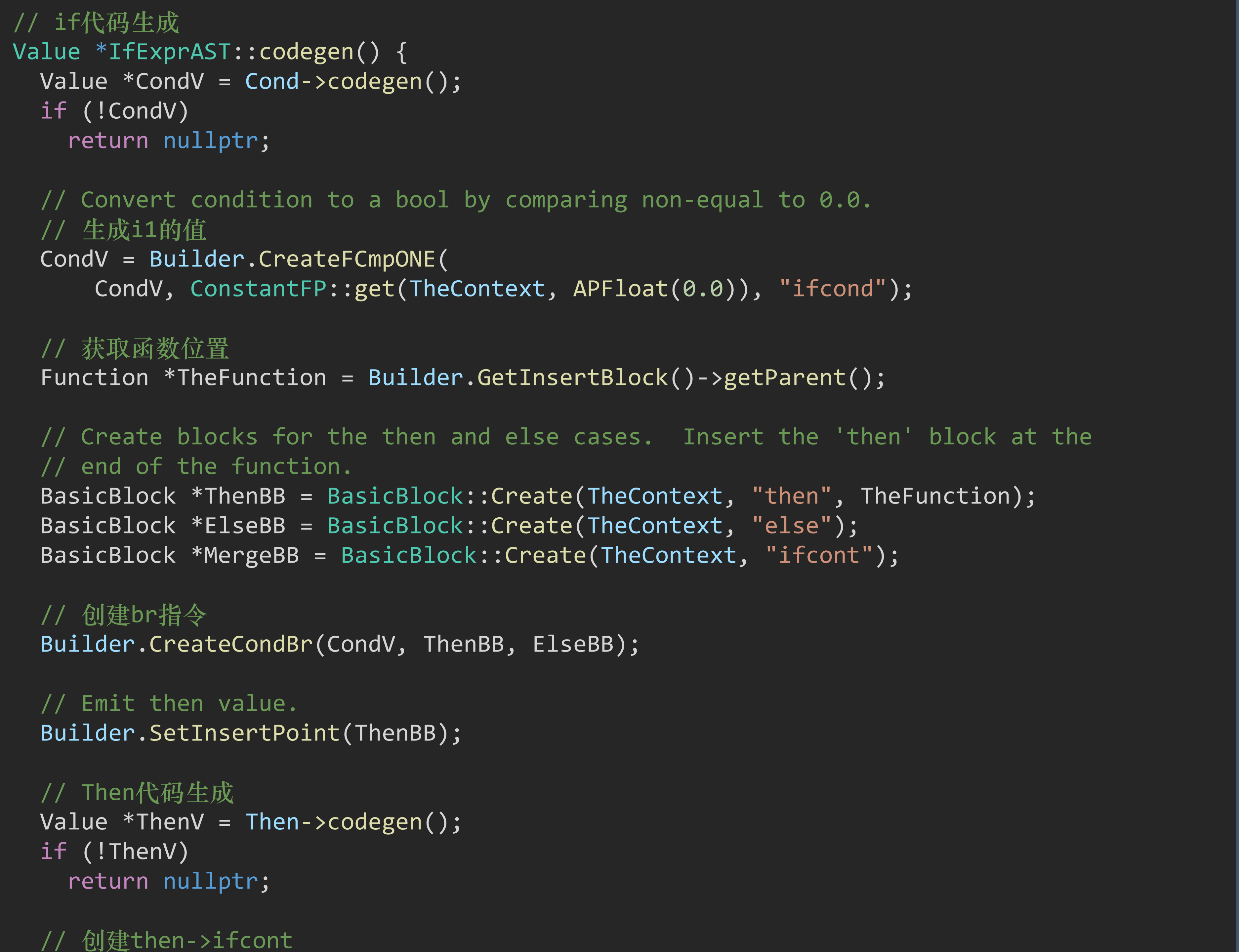


```
// 解析if的结构
static std::unique_ptr<ExprAST> ParseIfExpr() {
  getNextToken(); // eat the if.

  // condition.
  auto Cond = ParseExpression();
  if (!Cond)
    return nullptr;

  if (CurTok != tok_then)
    return LogError("expected then");
  getNextToken(); // eat the then

  auto Then = ParseExpression();
  if (!Then)
    return nullptr;

  if (CurTok != tok_else)
    return LogError("expected else");

  getNextToken();

  auto Else = ParseExpression();
  if (!Else)
    return nullptr;

  return std::make_unique<IfExprAST>(std::move(Cond), std::move(Then),
                                      std::move(Else));
}
```

for parser

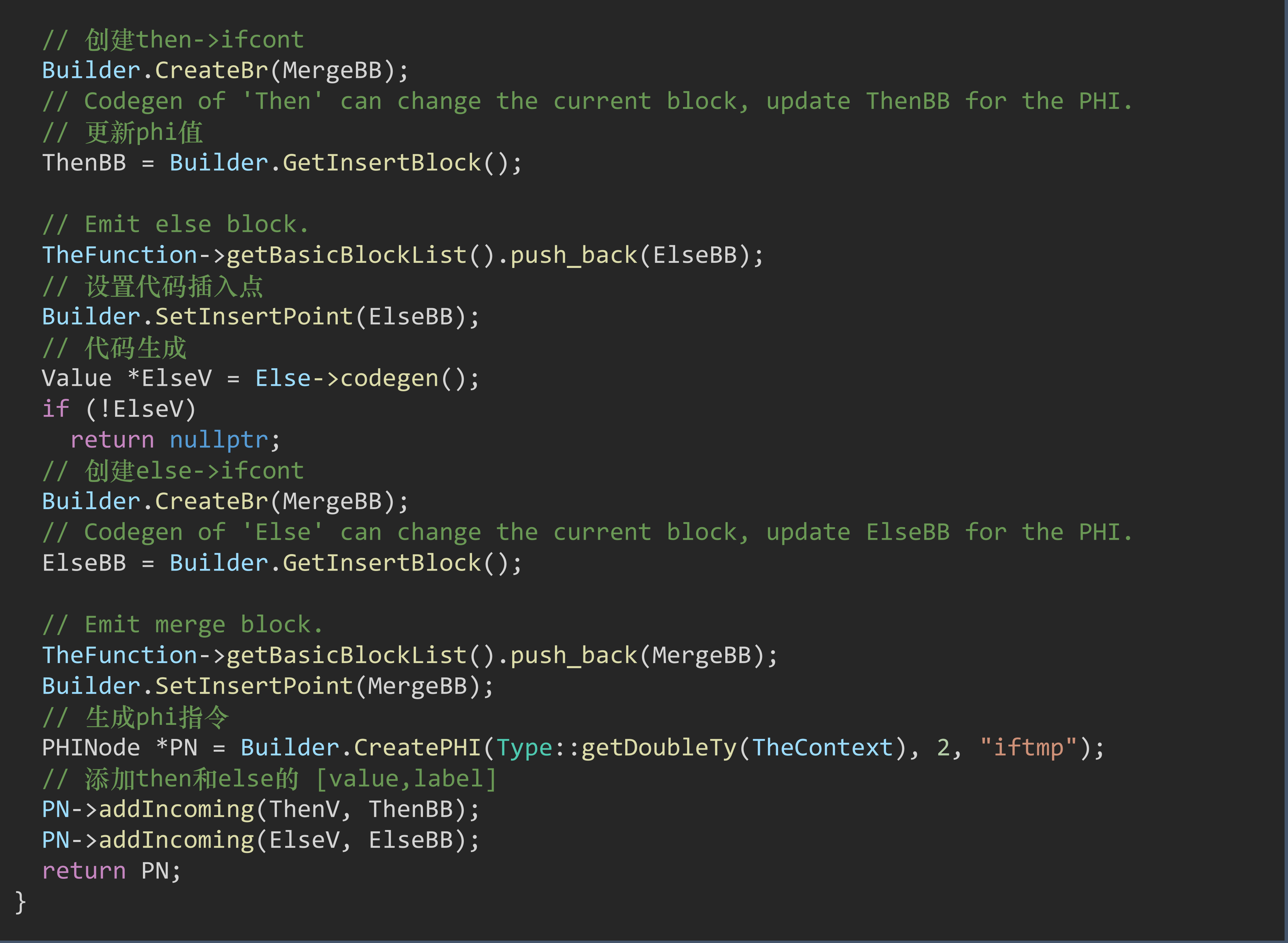
```
// 解析for的结构
static std::unique_ptr<ExprAST> ParseForExpr() {
  getNextToken(); // eat the for.

  if (CurTok != tok_identifier)
    return LogError("expected identifier after for");

  std::string IdName = IdentifierStr;
  getNextToken(); // eat identifier.

  if (CurTok != '=')
    return LogError("expected '=' after for");
  getNextToken(); // eat '='.

  auto Start = ParseExpression();
  if (!Start)
    return nullptr;
  if (CurTok != ',')
    return LogError("expected ',' after for start value");
  getNextToken();

  auto End = ParseExpression();
  if (!End)
    return nullptr;
    return LogError("expected ',' after for start value");
  getNextToken();

  auto End = ParseExpression();
  if (!End)
    return nullptr;

  // The step value is optional.
  std::unique_ptr<ExprAST> Step;
  if (CurTok == ',') {
    getNextToken();
    Step = ParseExpression();
    if (!Step)
      return nullptr;
  }

  if (CurTok != tok_in)
    return LogError("expected 'in' after for");
  getNextToken(); // eat 'in'.

  auto Body = ParseExpression();
  if (!Body)
    return nullptr;

  return std::make_unique<ForExprAST>(IdName, std::move(Start), std::move(End),
                                       std::move(Step), std::move(Body));
}
```

if 代码生成



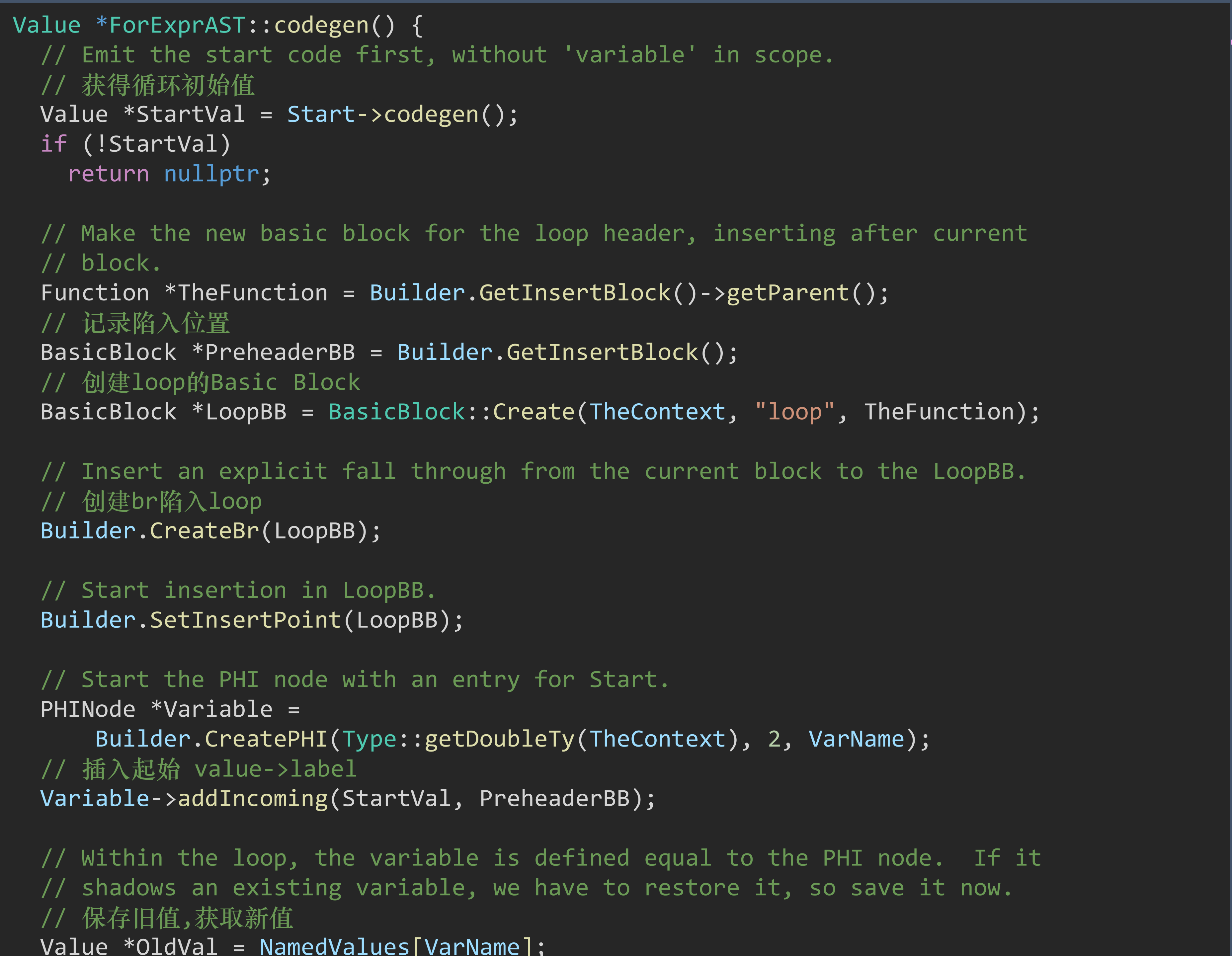
```
// if代码生成
Value *IfExprAST::codegen() {
  Value *CondV = Cond->codegen();
  if (!CondV)
    return nullptr;

  // Convert condition to a bool by comparing non-equal to 0.0.
  // 生成i1的值
  CondV = Builder.CreateFCmpONE(
      CondV, ConstantFP::get(TheContext, APFloat(0.0)), "ifcond");

  // 获取函数位置
  Function *TheFunction = Builder.GetInsertBlock()->getParent();

  // Create blocks for the then and else cases.  Insert the 'then' block at the
  // end of the function.
  BasicBlock *ThenBB = BasicBlock::Create(TheContext, "then", TheFunction);
  BasicBlock *ElseBB = BasicBlock::Create(TheContext, "else");
  BasicBlock *MergeBB = BasicBlock::Create(TheContext, "ifcont");

  // 创建br指令
  Builder.CreateCondBr(CondV, ThenBB, ElseBB);

  // Emit then value.
  Builder.SetInsertPoint(ThenBB);

  // Then代码生成
  Value *ThenV = Then->codegen();
  if (!ThenV)
    return nullptr;

  // 创建then->ifcont
```

if 代码生成



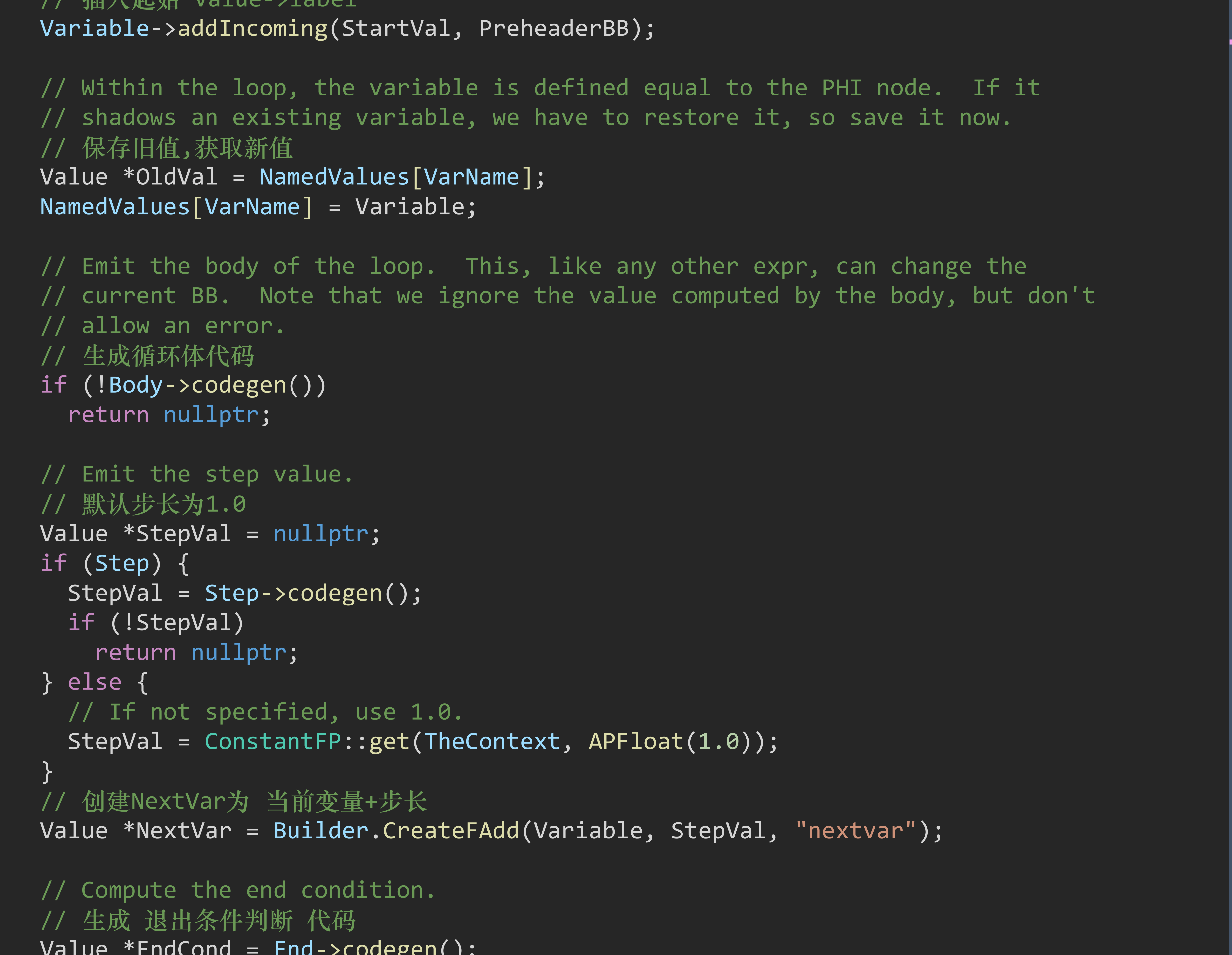
```
  // 创建then->ifcont
  Builder.CreateBr(MergeBB);
  // Codegen of 'Then' can change the current block, update ThenBB for the PHI.
  // 更新phi值
  ThenBB = Builder.GetInsertBlock();

  // Emit else block.
  TheFunction->getBasicBlockList().push_back(ElseBB);
  // 设置代码插入点
  Builder.SetInsertPoint(ElseBB);
  // 代码生成
  Value *ElseV = Else->codegen();
  if (!ElseV)
    return nullptr;
  // 创建else->ifcont
  Builder.CreateBr(MergeBB);
  // Codegen of 'Else' can change the current block, update ElseBB for the PHI.
  ElseBB = Builder.GetInsertBlock();

  // Emit merge block.
  TheFunction->getBasicBlockList().push_back(MergeBB);
  Builder.SetInsertPoint(MergeBB);
  // 生成phi指令
  PHINode *PN = Builder.CreatePHI(Type::getDoubleTy(TheContext), 2, "iftmp");
  // 添加then和else的 [value,label]
  PN->addIncoming(ThenV, ThenBB);
  PN->addIncoming(ElseV, ElseBB);
  return PN;
}
```

for代码生成



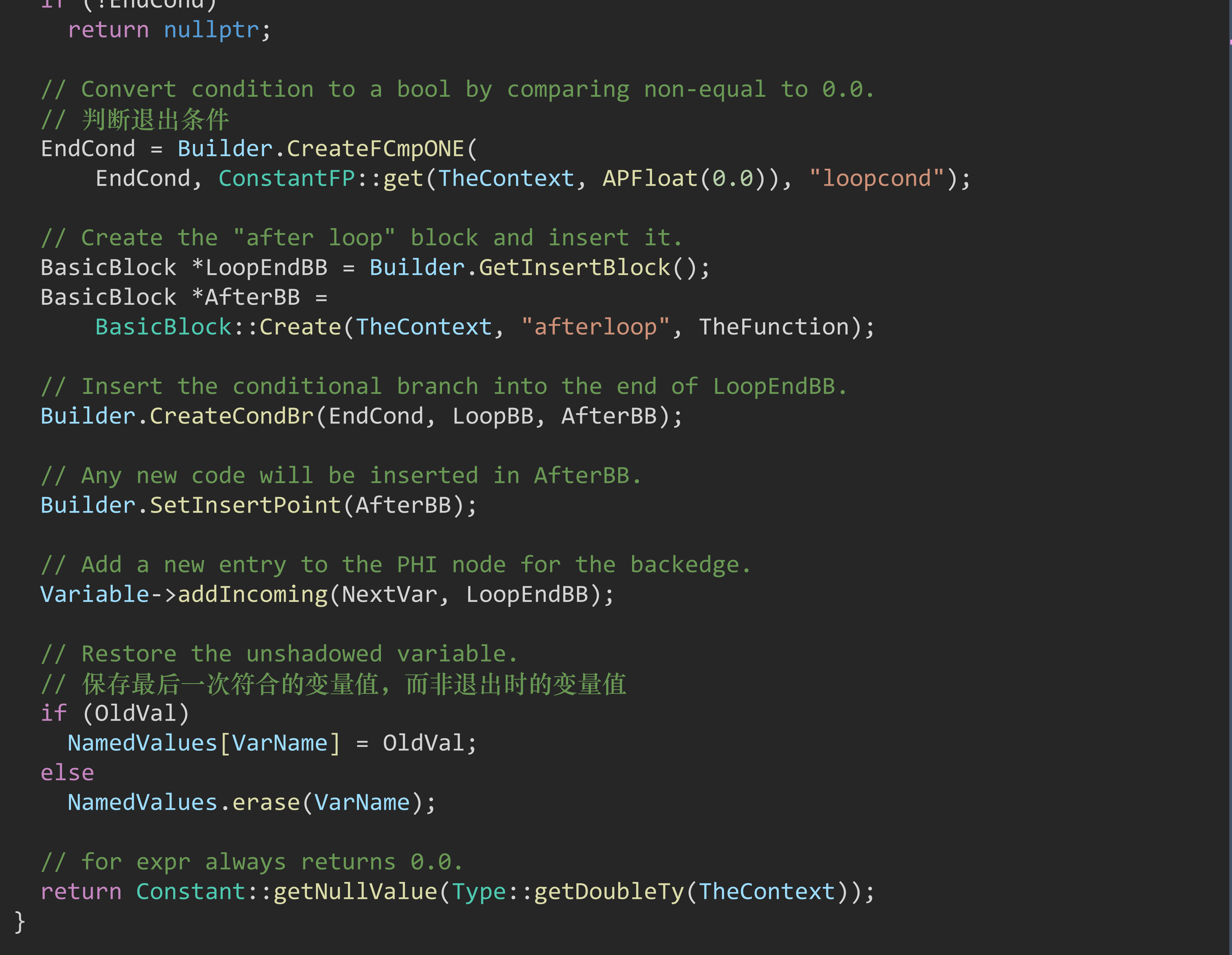
```
Value *ForExprAST::codegen() {
  // Emit the start code first, without 'variable' in scope.
  // 获得循环初始值
  Value *StartVal = Start->codegen();
  if (!StartVal)
    return nullptr;

  // Make the new basic block for the loop header, inserting after current
  // block.
  Function *TheFunction = Builder.GetInsertBlock()->getParent();
  // 记录陷入位置
  BasicBlock *PreheaderBB = Builder.GetInsertBlock();
  // 创建loop的Basic Block
  BasicBlock *LoopBB = BasicBlock::Create(TheContext, "loop", TheFunction);

  // Insert an explicit fall through from the current block to the LoopBB.
  // 创建br陷入loop
  Builder.CreateBr(LoopBB);

  // Start insertion in LoopBB.
  Builder.SetInsertPoint(LoopBB);

  // Start the PHI node with an entry for Start.
  PHINode *Variable =
      Builder.CreatePHI(Type::getDoubleTy(TheContext), 2, VarName);
  // 插入起始 value->label
  Variable->addIncoming(StartVal, PreheaderBB);

  // Within the loop, the variable is defined equal to the PHI node.  If it
  // shadows an existing variable, we have to restore it, so save it now.
  // 保存旧值,获取新值
  Value *OldVal = NamedValues[VarName];
```

for代码生成



```
// 插入起始 value->label
Variable->addIncoming(StartVal, PreheaderBB);

// Within the loop, the variable is defined equal to the PHI node.  If it
// shadows an existing variable, we have to restore it, so save it now.
// 保存旧值,获取新值
Value *OldVal = NamedValues[VarName];
NamedValues[VarName] = Variable;

// Emit the body of the loop.  This, like any other expr, can change the
// current BB.  Note that we ignore the value computed by the body, but don't
// allow an error.
// 生成循环体代码
if (!Body->codegen())
  return nullptr;

// Emit the step value.
// 默认步长为1.0
Value *StepVal = nullptr;
if (Step) {
  StepVal = Step->codegen();
  if (!StepVal)
    return nullptr;
} else {
  // If not specified, use 1.0.
  StepVal = ConstantFP::get(TheContext, APFloat(1.0));
}
// 创建NextVar为 当前变量+步长
Value *NextVar = Builder.CreateFAdd(Variable, StepVal, "nextvar");

// Compute the end condition.
// 生成 退出条件判断 代码
Value *EndCond = End->codegen();
```

for代码生成



```
  if (!EndCond)
    return nullptr;

  // Convert condition to a bool by comparing non-equal to 0.0.
  // 判断退出条件
  EndCond = Builder.CreateFCmpONE(
      EndCond, ConstantFP::get(TheContext, APFloat(0.0)), "loopcond");

  // Create the "after loop" block and insert it.
  BasicBlock *LoopEndBB = Builder.GetInsertBlock();
  BasicBlock *AfterBB =
      BasicBlock::Create(TheContext, "afterloop", TheFunction);

  // Insert the conditional branch into the end of LoopEndBB.
  Builder.CreateCondBr(EndCond, LoopBB, AfterBB);

  // Any new code will be inserted in AfterBB.
  Builder.SetInsertPoint(AfterBB);

  // Add a new entry to the PHI node for the backedge.
  Variable->addIncoming(NextVar, LoopEndBB);

  // Restore the unshadowed variable.
  // 保存最后一次符合的变量值，而非退出时的变量值
  if (OldVal)
    NamedValues[VarName] = OldVal;
  else
    NamedValues.erase(VarName);

  // for expr always returns 0.0.
  return Constant::getNullValue(Type::getDoubleTy(TheContext));
}
```

拓展阅读

- https://zh.wikipedia.org/wiki/静态单赋值形式