**第七章-定义变量和变量赋值设计文档**

一．定义变量Definition

1.Def语法树

返回变量名和内容，默认为空

class DefStatAST : public StatAST {

std::vector<std::string> VarNames;

std::unique\_ptr<StatAST> Body;

public:

DefStatAST(std::vector<std::string> VarNames, std::unique\_ptr<StatAST> Body)

:VarNames(std::move(VarNames)), Body(std::move(Body)) {}

Value \*codegen() override;

};

2.Def解析

首先初始化全局函数

static std::unique\_ptr<StatAST> ParseDefStat();

然后解析参照万花筒的定义函数

//解析变量声明

static std::unique\_ptr<StatAST> ParseDefStat() {

//eat 'VAR'

getNextToken();

std::vector<std::string> varNames;

//保证至少有一个变量的名字

if (CurTok != VARIABLE) {

return LogErrorD("expected identifier after VAR");

}

while (true)

{

varNames.push\_back(IdentifierStr);

//eat VARIABLE

getNextToken();

if (CurTok != ',')

break;

getNextToken();

if (CurTok != VARIABLE) {

return LogErrorD("expected identifier list after VAR");

}

}

auto Body = nullptr;

return llvm::make\_unique<DefStatAST>(std::move(varNames), std::move(Body));

}

3.IR代码生成

参照万花筒有

Value \*DefStatAST::codegen() {

std::vector<AllocaInst \*> OldBindings;

Function \*TheFunction = Builder.GetInsertBlock()->getParent();

for (unsigned i = 0, e = VarNames.size(); i != e; ++i) {

const std::string &VarName = VarNames[i];

Value \*InitVal = ConstantInt::get(TheContext, APInt(32, 0));

AllocaInst \*Alloca = CreateEntryBlockAlloca(TheFunction, VarName);

Builder.CreateStore(InitVal, Alloca);

OldBindings.push\_back(NamedValues[VarName]);

NamedValues[VarName] = Alloca;

}

return nullptr;

}

同时我们将更改NamedValues映射，使其映射到AllocaInst \*而不是Value \*。

//static std::map<std::string, Value \*> NamedValues;

static std::map<std::string, AllocaInst \*> NamedValues;

并定义辅助函数

// CreateEntryBlockAlloca - Create an alloca instruction in the entry block of

// the function. This is used for mutable variables etc.

static AllocaInst \*CreateEntryBlockAlloca(Function \*TheFunction,

const std::string &VarName) {

IRBuilder<> TmpB(&TheFunction->getEntryBlock(),

TheFunction->getEntryBlock().begin());

return TmpB.CreateAlloca(Type::getInt32Ty(TheContext), nullptr,

VarName.c\_str());

}

二.变量赋值

一开始参照万花筒考虑将“:=”作为一个运算符进行操作，但是因为这样的时候Op保存两个字符，因为getNextToken的机制为一次读一个，实际进行操作的时候始终没法完美的实现赋值语句，于是最终决定建立一个新函数赋值读取ASSIGN\_SYMBOL来完成赋值操作。

1.Assign语法树

class AssStatAST : public StatAST {

std::unique\_ptr<VariableExprAST> Name;

std::unique\_ptr<StatAST> Expression;

public:

AssStatAST(std::unique\_ptr<VariableExprAST> Name, std::unique\_ptr<StatAST> Expression)

: Name(std::move(Name)), Expression(std::move(Expression)) {}

Value \*codegen() override;

};

返回变量名并将后方的操作作为一个表达式存储并返回。

2.Assign解析

同样先初始化全局函数

static std::unique\_ptr<StatAST> ParseAssStat();

然后解析有

//解析 赋值语句

static std::unique\_ptr<StatAST> ParseAssStat() {

auto a = ParseIdentifierExpr();

VariableExprAST\* Name = (VariableExprAST\*)a.get();

auto NameV = llvm::make\_unique<VariableExprAST>(Name->getName());

if (!Name)

return nullptr;

if (CurTok != ASSIGN\_SYMBOL)

return LogError("need := in assignment statment");

getNextToken();

auto Expression = ParseExpression();

if (!Expression)

return nullptr;

return llvm::make\_unique<AssStatAST>(std::move(NameV), std::move(Expression));

}

3.IR生成

Value \*AssStatAST::codegen() {

Value\* EValue = Expression->codegen();

if (!EValue)

return nullptr;

Value \*Variable = NamedValues[Name->getName()];

if (!Variable)

return LogErrorV("Unknown variable name");

Builder.CreateStore(EValue, Variable);

return EValue;

}

将赋值符后方的数字和变量名存储在NamedValues作为一个新的变量名

三．将两个语句添加到statement里分析做两个新的case