编译原理研讨课实验PR001实验报告

一、任务说明

本次实验的任务为：

1.熟悉Clang的安装和使用

1）掌握如何从源代码编译安装LLVM和Clang

2）了解如何生成和查看C程序对应的AST

2.通过修改前端#pragma 到AST的信息传递，使编译器能够识别#pragma elementWise，并在AST中增加相应元素。

对于添加了制导的源程序 \*.c ，按照规则在编译时打印每个函数的名称，该函数是否在制导范围内。 对于一个函数的是否在制导范围内的定义:

1） 区域以函数/过程的定义（不是声明）为单位

2） 一个制导总是匹配在其后出现的，离它最近的一个函数定义

3） 一个制导只能匹配最多一个函数定义

二、成员组成

李昊宸 2017K8009929044

李颖彦 2017K8009929025

陆润宇 2017K8009929027

三、实验设计

将实验分为：parse（语法解析），sema（语义），AST（语法树）三个部分来逐一实现。

对于给定的代码框架，采用gdb跟随clang基础部分以及TraverseFunctionDecls.cpp运行时的函数调用流程，可以定位代码框架中为识别#pragma elementWise而需要进行补充的地方，进而完成实验所需的内容。

四、设计思路

熟悉Clang的安装和使用：

1. 准备一个C程序 test.c

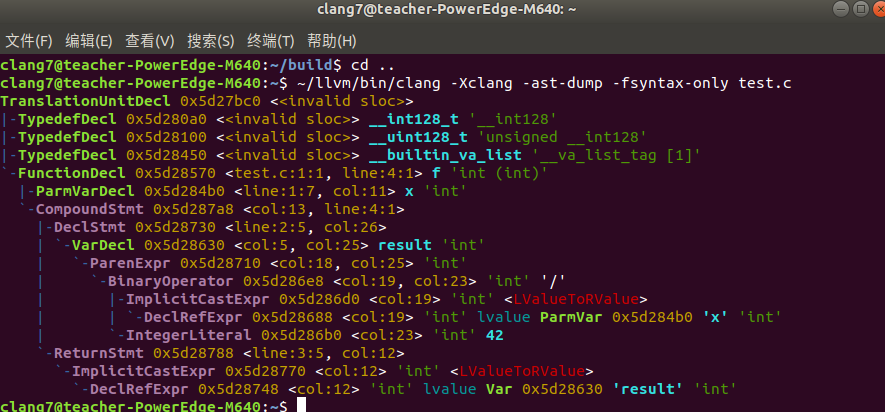
int f(int x) {

int result = (x / 42);

return result;

}

2. 使用`clang`将AST给dump出来：`~/llvm-install/bin/clang -Xclang -ast-dump -fsyntax-only test.c`



图一 test.c的AST

添加element wise的制导：

1. 语法阶段

首先通过parse的构造函数来对elementWise的PragmaHandler进行初始化。之后，设计在HandlerPragma()中被调用的PragmaElementWiseHandler()，将token赋为annot\_pragma\_elementWise的类型，重新放回token流中。

之后，在对Declaration的解析当中，依照token类型annot\_pragma\_elementWise，调用HandlerPragmaElementWise()函数进行处理，它再调用ActOnPragmaElementWise()函数，目的是对elementWise进行实质上的功能实现。

2. 语义阶段

在prj1中，我们只需要实现当#pragma elementWise出现时，将对应的函数打印成1并输出，所以ActOnPragmaElementWise()的内容即为ElementWiseContext置1。

3. 构建AST阶段

为每个函数声明新增一个变量isElementWise，每当一个函数定义时，若ElementWiseContext的值为1，则将这个定义内部的isElementWise变量置为1，再将ElementWiseContext清零。这样，在输出插件遍历函数声明时，便可以判断具体函数是否是elememtWise的。

五、实验实现

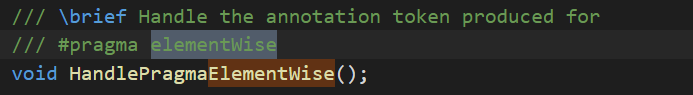
1. 语法阶段

/home/clang7/llvm/tools/clang/include/clang/Parse/Parser.h

借助模板定义ElementWiseHandler

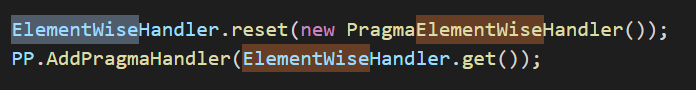


在class parse中声明：

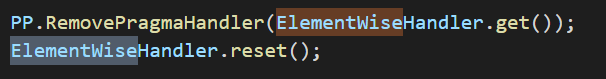


/home/clang7/llvm/tools/clang/lib/Parse/Parser.cpp

在parse的构造函数中，对elementwise的PragmaHandler进行初始化



这里注册了PragmaElementWiseHandler，使下面的FindHandler函数可以根据token的name来找到PragmaElementWiseHandler的入口地址



这里移除elementwise的pragmahandler并重置

/home/clang7/llvm/tools/clang/lib/Lex/Pragma.cpp

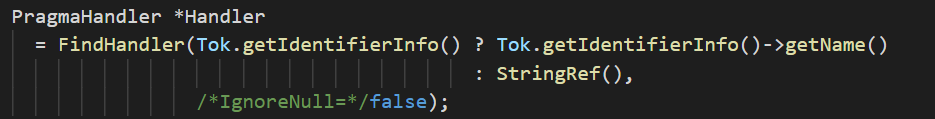
在函数

void PragmaElementWiseHandler::HandlePragma(Preprocessor &PP,

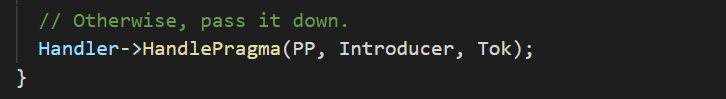
PragmaIntroducerKind Introducer,

Token &ElementWiseTok)

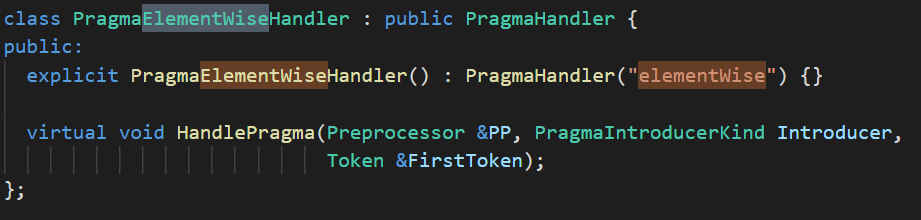
中通过调用



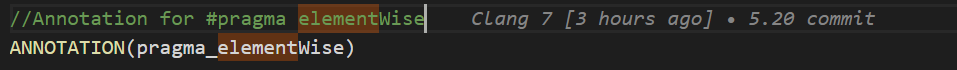
发现handler为PragmaElementWiseHandler, 并进行调用

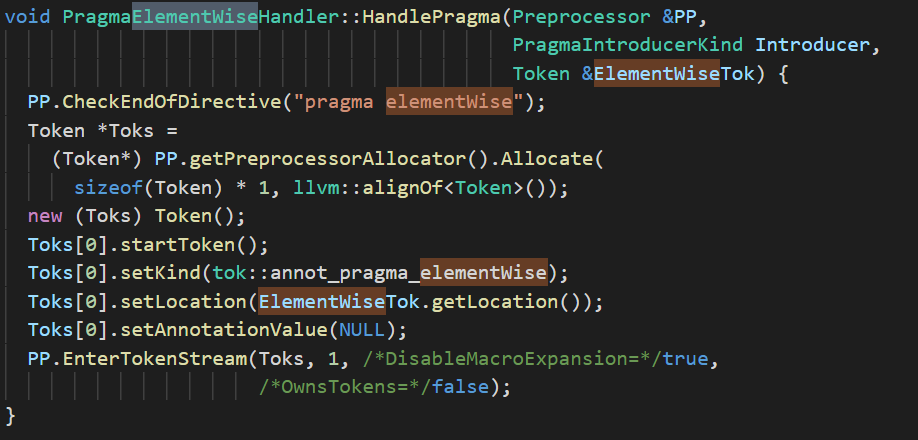


/home/clang7/llvm/tools/clang/lib/Parse/ParsePragma.h



/home/clang7/llvm/tools/clang/include/clang/Basic/TokenKinds.def



/home/clang7/llvm/tools/clang/lib/Parse/ParsePragma.cpp

这里调用了

/home/clang7/llvm/tools/clang/lib/Lex/PPDirectives.cpp

中的

void Preprocessor::CheckEndOfDirective(const char \*DirType, bool EnableMacros)

检查后缀pragma elementwise

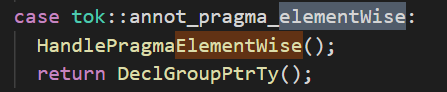
并通过

Toks[0].setKind(tok::annot\_pragma\_elementWise);

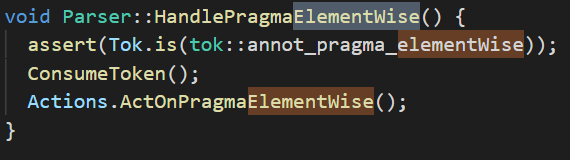
设置token的类型为annot\_pragma\_elementWise

/home/clang7/llvm/tools/clang/lib/Parse/Parser.cpp

在Parser::ParseExternalDeclaration函数中，根据tok的不同类型来选择调用相应的函数，如果tok是elementwise的类型，则调用HandlePragmaElementWise()



/home/clang7/llvm/tools/clang/lib/Parse/ParsePragma.cpp



调用Sema阶段的函数ActOnPragmaElementWise

2. 语义阶段

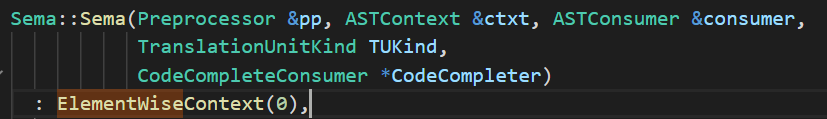
/home/clang7/llvm/tools/clang/include/clang/Sema/Sema.h





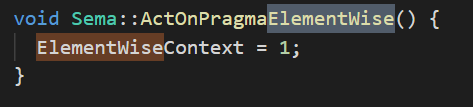
/home/clang7/llvm/tools/clang/lib/Sema/Sema.cpp

在sema的构造函数中，将ElementWiseContext的初始值赋值为0



/home/clang7/llvm/tools/clang/lib/Sema/SemaAttr.cpp

ActOnPragmaElementWise()的功能为被调用时将ElementWiseContext赋值为1



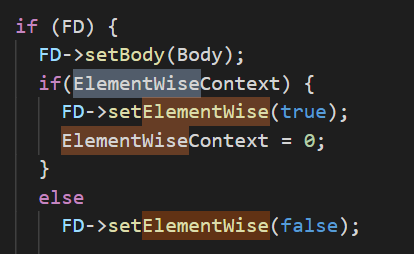
/home/clang7/llvm/tools/clang/lib/Sema/SemaDecl.cpp

在函数

Decl \*Sema::ActOnFinishFunctionBody(Decl \*dcl, Stmt \*Body,

bool IsInstantiation)

中有



如果ElementWiseContext为1，则setElementWise函数会将IsElementWise赋值为true，并将ElementWiseContext清零；否则IsElementWise置为false

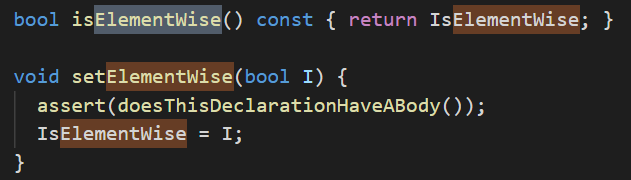
3. AST

/home/clang7/llvm/tools/clang/include/clang/AST/Decl.h

声明IsElementWise的位域为1：



构建setElementWise函数



最后，在输出插件

/home/clang7/llvm/tools/clang/examples/TraverseFunctionDecls/TraverseFunctionDecls.cpp中，根据isElementWise的值来决定相应的输出





4. 主要调用流程图

startToken

LexUnexpandedToken

LexTokenInternal

Lex

（Lexer）

Lex（Preprocessor）

CheckEndOfDirective

FindHandler

HandlePragma

Allocate

PragmaElementWiseHandler::

HandlePragma

Token

ParseExternalDeclaration

startToken

setKind

HandlePragmaElementWise

setLocation

ActOnPragmaElementWise

setAnnotation

Value

ActOnFinish

FunctionBody

EnterTokenStream

六、总结

本次实验主要是围绕clang的词法和语法方面，首先识别#pragma elementWise，并调用相关函数来对token进行处理。处理之后，将变量ElementWiseContext置1。随后AST处理时若ElementWiseContext为1，则将该函数声明的内部值IsElementWise置1，将ElementWiseContext清零。如此即可实现对于elementWise的检测和标识。

本次实验成功实现了任务要求，且代码书写切近源码风格，较为清晰。

以下是几组个人测试结果：

测试一：



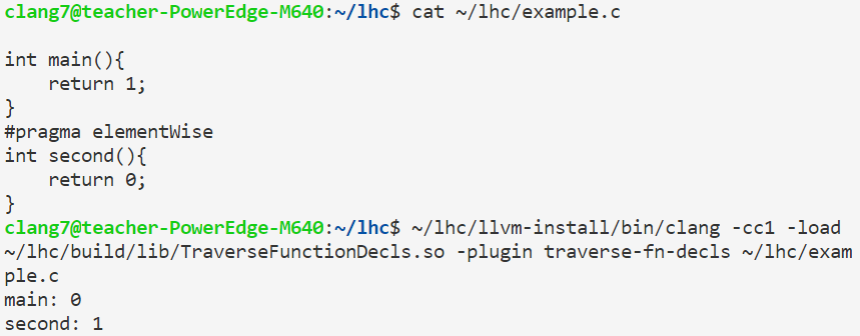
正常情况下，制导范围包括第一个遇到的函数定义main，不包括second。

测试二：



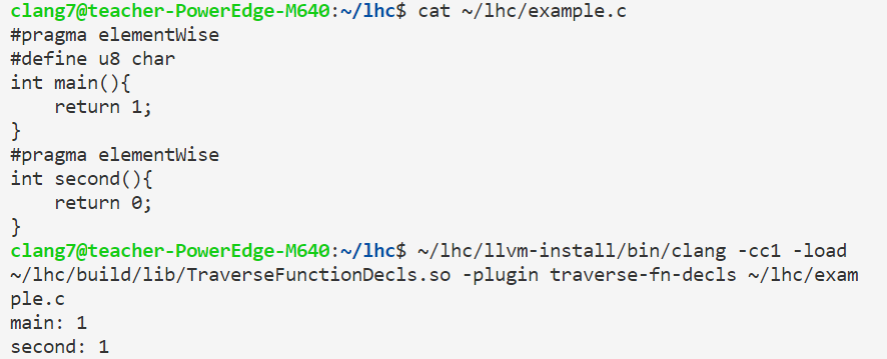
正常情况下，第一个制导范围包括第一个遇到的函数定义main，第二个制导范围包括second。

测试三：



正常情况下，制导范围包括第一个遇到的函数定义second，不包括之前的函数定义main。

测试四：



制导与函数定义间加入其他语句（如宏定义），不影响对函数定义的制导

测试五：



制导与函数定义之间增加函数声明，制导后第一个遇到的函数定义second在制导范围内，函数声明foo并不在制导范围内。

七、成员总结

李昊宸：本次实验负责clang代码的书写，以及实验报告的修改和补充。第一次处理大工程代码，起初无从下手，还好老师给出了处理pack制导的样例，得以比较快的了解clang代码的运行过程，并且实现了仿照pack完成的element wise制导定义。每个调用的函数名都较为直观，可以快速理解函数实现的功能，但是在不同的分析过程下，经常会出现同名函数（但是不在同一名字空间），导致起初较难理解调用过程。

李颖彦：

本次实验负责实验报告的书写。因为本次实验是第一次接触clang源代码，所以理解clang源码的结构和功能便花了很长的时间，同时还需要有一定C++的面向对象的基础。前期的准备工作做的很长。在理解了clang源码基础上，实验则变得简单了很多。

陆润宇：

本次实验负责实验报告的修改和补充。在clang源代码中，用到了C++类相关的功能，与我之前进行的其他实验相比，我觉得这种方式屏蔽了更多底层的细节，使得代码更加清晰明了。在这次实验中，使用gdb跟踪运行起到了很关键的作用，这使我们能够沿着程序运行轨迹逐步补充需要的内容，减少了不必要的代码阅读量，也帮助我们有效地理解了clang代码的部分运行机制。