PR001 实验报告

第 11 组 谭弘泽 张传奇 冯凌璇

【实验目标】

在程序中添加航天子集区域限制的制导#pragma asCheck，区域以函数/过程为单位。

实验最终形式如下：

输入：添加制导的源程序*.c

输出：打印每个函数对应的是否在制导范围内

输出格式为：

函数名:1 或者 0（1 表示有 asCheck 制导，0 表示没有）

每个函数一行

【实验流程】

1. 考虑四种会报 warning 的情况
   a) #pragma asCheck 后面跟着的不是函数的定义。
      `"expected function defination after pragma asCheck - ignoring"`
   b) 遇到 eof 时，#pragma asCheck 后面的记号与它不在同一个文件中。
      `"pragma asCheck should followed by a function defination but meet end of file - ignoring"`
   c) #pragma asCheck 与 eod 之间有其他东西。
      `"pragma asCheck should stand alone without following components - ignoring"`
   d) 非语法错误的其他错误。
      `"pragma asCheck should be declare in front of a function defination - ignoring"`
2. llvm 对 asCheck 制导的处理
   a) 编译器读到#pragma asCheck 时首先调用 HandlePragma 函数，该函数会不停读记号直到读到函数的制导语句结束标记 “eod” 为止，同时用 i 对读到的记号进行计数。若 i>0 表示在#pragma asCheck 与 eod 之间存在其他记号，这种情况被认为是非法的，并报 warning:"pragma asCheck should stand alone without following components - ignoring"。接着，在这个函数中会声明一个种类为 “annot_pragma_ascheck” 的

记号并且将这个记号插入到记号流中。

b) 之后读到这个 token 的时候会通过 token 的种类判断是“annot_pragma_ascheck”，并且触发 HandlePragmaAsCheck 函数。在该函数中首先向前看一个记号，得到 eod 下面一个记号的位置 ahd，比较 ahd 的位置和 Tok 的位置对应的文件 ID 可以解决跨文件的问题。如果文件 ID 不同会报 warning: "pragma asCheck should followed by a function defination but meet end of file - ignoring"。接着会根据 ahd 的位置执行 ActOnPragmaAsCheck 函数，得到函数定义在文件中的偏移量并加入表中。最后 consume 掉 eod。

c) 之后读到一个函数定义 FD 时会调用 ActOnPendingAsCheck 函数。该函数会比较表中的定义与该函数定义是否是同一个，如果能找到说明 FD 需要做 AsCheck，于是将其 setAsCheck，否则说明 AsCheck 后面没有跟着一个函数定义，于是报 warning: "expected function definition after pragma asCheck - ignoring"并将当前表中内容都删除。

d) 最后读到 eof 时调用 ActOnDropingAsCheck 函数，检查表中是否还有剩余 AsCheck，如果有则将其全部删除并报 warning: "expected function definition after pragma asCheck - ignoring"。

3. 关键函数（代码）实现

a) tools/clang/include/clang/AST/Decl.h

在 FunctionDecl 中添加变量 HasAsCheck，用于标记 AsCheck。

添加相应的读取和设置函数 isAsCheck 和 setAsCheck。

```
bool isAsCheck() const { return HasAsCheck; }
void setAsCheck(bool asCheck) { HasAsCheck = asCheck; }
```

b) tools/clang/include/clang/Parser/Parser.h

添加助手函数 OwningPtr<PragmaHandler> AsCheckHandler。

c) tools/clang/include/clang/Parser/ParserPragma.h

增加处理#pragma asCheck 的结构

```
struct PragmaAsCheckHandler : public PragmaHandler {
  PragmaAsCheckHandler() : PragmaHandler("asCheck") {}
  virtual void HandlePragma(Preprocessor &PP,
      PragmaIntroducerKind Introducer,
      Token &Tok);
};
```

d) tools/clang/lib/Sema/SemaDecl.cpp

增加三个函数 ActOnPragmaAsCheck、ActOnPendingAsCheck、ActOnDropingAsCheck。具体作用如上节所述。

```
void Sema::ActOnPragmaAsCheck(SourceLocation Loc){
  std::pair<FileID,unsigned> deLoc =
getSourceManager().getDecomposedLoc(Loc);
  AsCheckML[deLoc.first].push_back(deLoc.second);
}


void Sema::ActOnPendingAsCheck(FunctionDecl* FD)
{
  if(FD){
    FD->setAsCheck(false);
    std::pair<FileID,unsigned> ps =
getSourceManager().getDecomposedLoc(FD->getLocStart());
    std::list<unsigned>* Locs = &AsCheckML[ps.first];
    {
        for(std::list<unsigned>::iterator i = Locs->begin() ; i !=
Locs -> end() ; Locs->erase(i++) )
        {
                FD->setAsCheck(true);
      }
      else
      {
    Diag(getSourceManager().getLocForStartOfFile(ps.first).getLocW
ithOffset(*i),diag::warn_pragma_ascheck_expected_func_def);
      }
        }
    }
  }
}
void Sema::ActOnDropingAsCheck(SourceLocation loc)
{
```

```
    for( std::map<FileID,std::list<unsigned> >::iterator kvpair =
AsCheckML.begin(); kvpair != AsCheckML.end() ; kvpair++)
    {
        std::list<unsigned>* Locs = &kvpair->second;
        for(std::list<unsigned>::iterator i = Locs->begin() ; i !=
Locs -> end() ; Locs->erase(i++) )
        {
      Diag(getSourceManager().getLocForStartOfFile(kvpair->first).
getLocWithOffset(*i),diag::warn_pragma_ascheck_expected_func_def);
        }
    }
}
```

在 ActOnStartOfFunctionDef 函数中，添加对 ActOnPendingAsCheck 函数的调用。

e) tools/clang/lib/Parser/ParserPragma.cpp

添加 AsCheckHandle 和 HandlePragma 的实现

```
void Parser::HandlePragmaAsCheck(){
  assert(Tok.is(tok::annot_pragma_ascheck));

  int forward = 1;
  const Token *ahd = &GetLookAheadToken(forward);
  static int i=0;
  i++;
  while(ahd->isNot(tok::eof)&&!ahd->getLocation().isValid())
  {
    ahd = &GetLookAheadToken(++forward);
  }

  std::pair<FileID,unsigned> ahdLoc =
getPreprocessor().getSourceManager().getDecomposedLoc(ahd->getLoca
tion());
```

```
  std::pair<FileID,unsigned> curLoc =
getPreprocessor().getSourceManager().getDecomposedLoc(Tok.getLocat
ion());
  if(curLoc.first!=ahdLoc.first)
  {
    Diag(Tok,diag::warn_pragma_ascheck_unexpected_eof);
  }
  else Actions.ActOnPragmaAsCheck(ahd->getLocation());
  ConsumeToken();//Consume eod
}

void PragmaAsCheckHandler::HandlePragma(Preprocessor &PP,
                                        PragmaIntroducerKind
Introducer,
                                        Token &Tok) {
    int i  = 0;
    while(true)
    {
        PP.LexUnexpandedToken(Tok);
        if(Tok.is(tok::eod))break;
        i++;
    }
    if(i>0)
    {
        PP.Diag(Tok,diag::warn_pragma_ascheck_unexpected_token);
        return ;
    }

    Token *Toks = (Token*) PP.getPreprocessorAllocator().Allocate(
        sizeof(Token) * 1, llvm::alignOf<Token>());
    new (Toks) Token();
    Toks[0].startToken();
    Toks[0].setKind(tok::annot_pragma_ascheck);
```

```
    Toks[0].setLocation(Tok.getLocation());
    PP.EnterTokenStream(Toks, 1, /*DisableMacroExpansion=*/true,
                        /*OwnsTokens=*/false);
}
```

f) tools/clang/lib/Parser/Parser.cpp

在其中增加一个“annot_pragma_ascheck”的case从而触发HandlePragmaAsCheck函数。

```
  case tok::annot_pragma_ascheck:
    HandlePragmaAsCheck();
    return DeclGroupPtrTy();
```

g) tools/clang/include/clang/Basic/TokenKinds.def

添加 ANNOTATION(pragma_ascheck)

h) tools/clang/include/clang/Basic/DiagnosticParseKinds.td

增加 warning 信息

```
def warn_pragma_ascheck_expected_func_def : Warning<
  "expected function defination after pragma asCheck - ignoring">;
def warn_pragma_ascheck_unexpected_eof : Warning<
  "pragma asCheck should followed by a function defination but
meet end of file - ignoring">;
def warn_pragma_ascheck_unexpected_location : Warning<
  "pragma asCheck should be declare in front of a function
defination - ignoring">;
def warn_pragma_ascheck_unexpected_token : Warning<
  "pragma asCheck should stand alone without following components
- ignoring">;
```

i) tools/calng/examples/PrintFunctionNames.cpp

用于打印结果

```
using namespace clang;

namespace {

class PrintFunctionsConsumer : public ASTConsumer {
public:
```

```
  virtual bool HandleTopLevelDecl(DeclGroupRef DG) {
    for (DeclGroupRef::iterator i = DG.begin(), e = DG.end(); i !=
e; ++i) {
      const Decl *D = *i;
      const NamedDecl *ND = dyn_cast<NamedDecl>(D);
      const FunctionDecl *FD = dyn_cast<FunctionDecl>(D);
      if (ND&&FD&&FD->isThisDeclarationADefinition())
      {
        llvm::errs() << ND->getNameAsString() <<":" <<
FD->isAsCheck() << "\n";
      }
    }

    return true;
  }
};

class PrintFunctionNamesAction : public PluginASTAction {
protected:
  ASTConsumer *CreateASTConsumer(CompilerInstance &CI,
llvm::StringRef) {
    return new PrintFunctionsConsumer();
  }

  bool ParseArgs(const CompilerInstance &CI,
                 const std::vector<std::string>& args) {
    for (unsigned i = 0, e = args.size(); i != e; ++i) {
      llvm::errs() << "PrintFunctionNames arg = " << args[i] <<
"\n";

      // Example error handling.
      if (args[i] == "-an-error") {
        DiagnosticsEngine &D = CI.getDiagnostics();
```

```
        unsigned DiagID = D.getCustomDiagID(
          DiagnosticsEngine::Error, "invalid argument '" + args[i]
+ "'");
        D.Report(DiagID);
        return false;
      }
    }
    if (args.size() && args[0] == "help")
      PrintHelp(llvm::errs());

    return true;
  }
  void PrintHelp(llvm::raw_ostream& ros) {
    ros << "Help for PrintFunctionNames plugin goes here\n";
  }

};

}

static FrontendPluginRegistry::Add<PrintFunctionNamesAction>
X("print-fns", "print function names");
```

4. 修改文件汇总

CMakeLists.txt examples
PrintFunctionNames.cpp examples/PrintFunctionNames
Decl.h include/clang/AST
DiagnosticParseKinds.td include/clang/Basic
TokenKinds.def include/clang/Basic
Parser.h include/clang/Parse
Sema.h include/clang/Sema
ParsePragma.cpp lib/Parse
ParsePragma.h lib/Parse
Parser.cpp lib/Parse
ParseStmt.cpp lib/Parse
SemaDecl.cpp lib/Sema

【实验结果】

1. 测试样例详见 testCases 文件夹，其中共包含 11 个测试文件

   1

   2

   3

   CE1

   CE2

   CE3

   CE4

   trival00

   trival01

   trival10

   trival11

1. 平凡测试

   a) 测试一

```
int foo()
{
    return 1;
}
```

```
int aaa
;
int
main()
{
    {
        foo();
    }
    return 0;
}
```

测试结果：

foo:0

main:0

b) 测试二

```
int foo()
{
    return 1;
}
int aaa
;
#pragma asCheck
int
main()
{
    {
        foo();
    }
    return 0;
}
```

测试结果：

foo:0

main:1

c) 测试三

```
#pragma asCheck
int foo()
{
    return 1;
}
int aaa
;
int
main()
{
    {
        foo();
    }
    return 0;
}
```

测试结果：

foo:1

main:0

d) 测试四

```
#pragma asCheck
int foo()
{
    return 1;
}
int aaa
;
#pragma asCheck
int
main()
{
    {
        foo();
    }
```

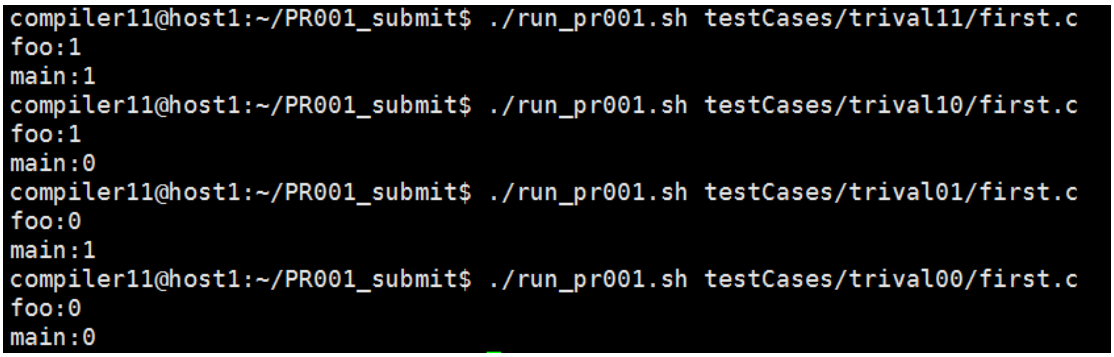
```
    return 0;
}
```

测试结果：

foo:1

main:1

测试截图：



2. 特殊情况测试

a) 测试一

一个文件

```
int foo()
{
    return 1;
}
#pragma asCheck
int aaa
;
#pragma asCheck
#pragma asCheck
int
main()
{
    #pragma asCheck
    {
        printf("hello world\n");
    }
    return 0;
```

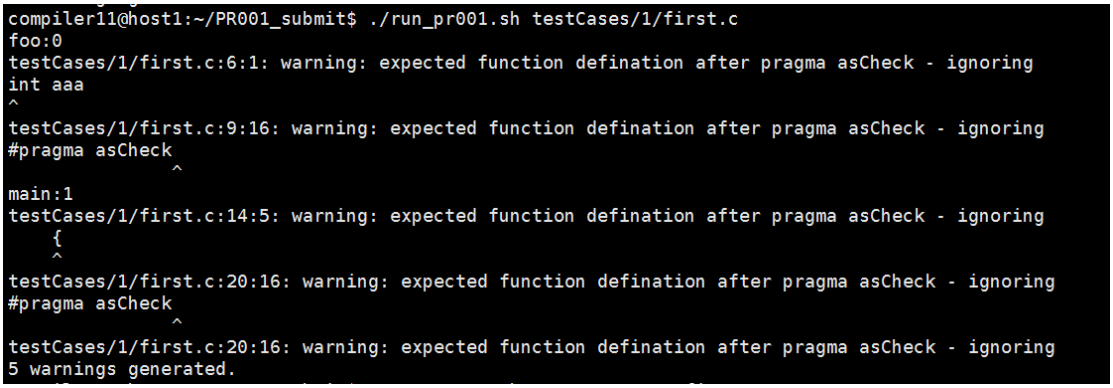
```
}
#pragma asCheck
#pragma asCheck
```

测试结果：

foo:0

main:1

测试截图：



b) 测试二

两个文件：

first.c

```
#include "second.c"
int foo()
{
    return 1;
}
int aaa
;
int
main()
{
    int i=0;
    for(i=0;i<0; i++)
    {
          int j = 0;
        #pragma asCheck
        {
```

```
        }
    }
    return 0;
}
#pragma asCheck
```

second.c

```
#pragma asCheck
int woo()
{
    return 8;
}
int foo();
#pragma asCheck
```

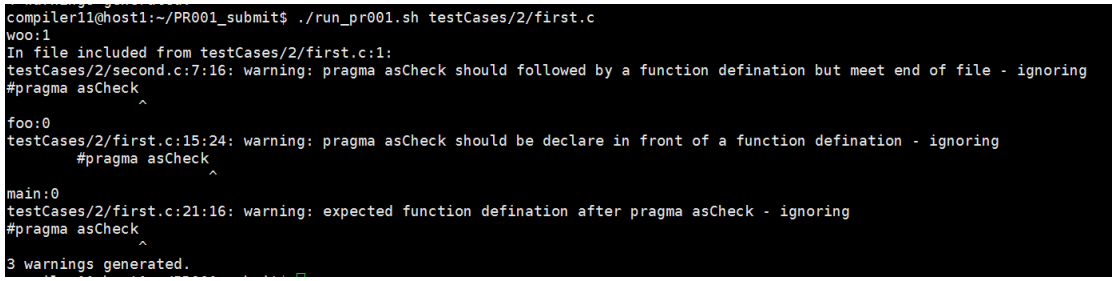
测试结果：

woo:1

foo:0

main:0

测试截图：



c) 测试三

三个文件

first.c

```
#include "second.c"
#include "third.c"
int foo()
{
    return 1;
}
int aaa
```

```
;
int
main()
{
    int i=0;
    for(i=0;i<0; i++)
    {
          int j = 0;
        #pragma asCheck
        {
        }
    }
    return 0;
}
```

second.c

```
#pragma asCheck
int woo()
{
    return 8;
}
#pragma asCheck
int foo();
```

third.c

```
int yoo()
{
    return 8;
}
#pragma asCheck
```

测试结果：

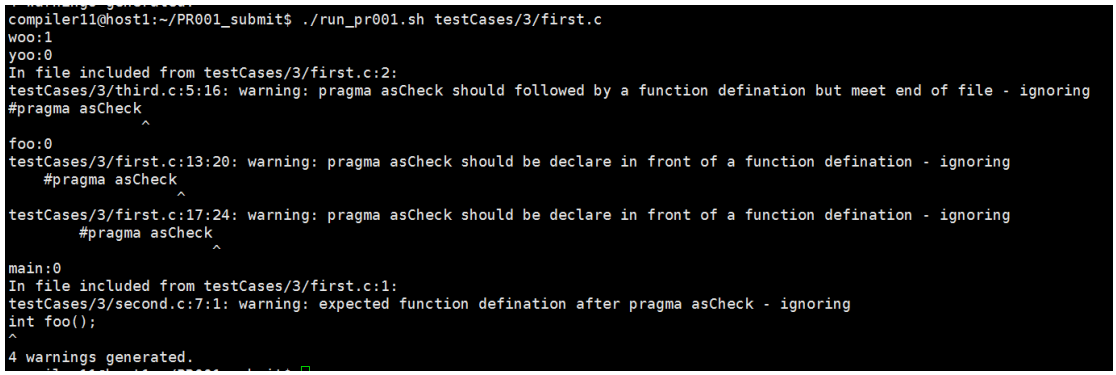
woo:1

yoo:0

foo:0

main:0

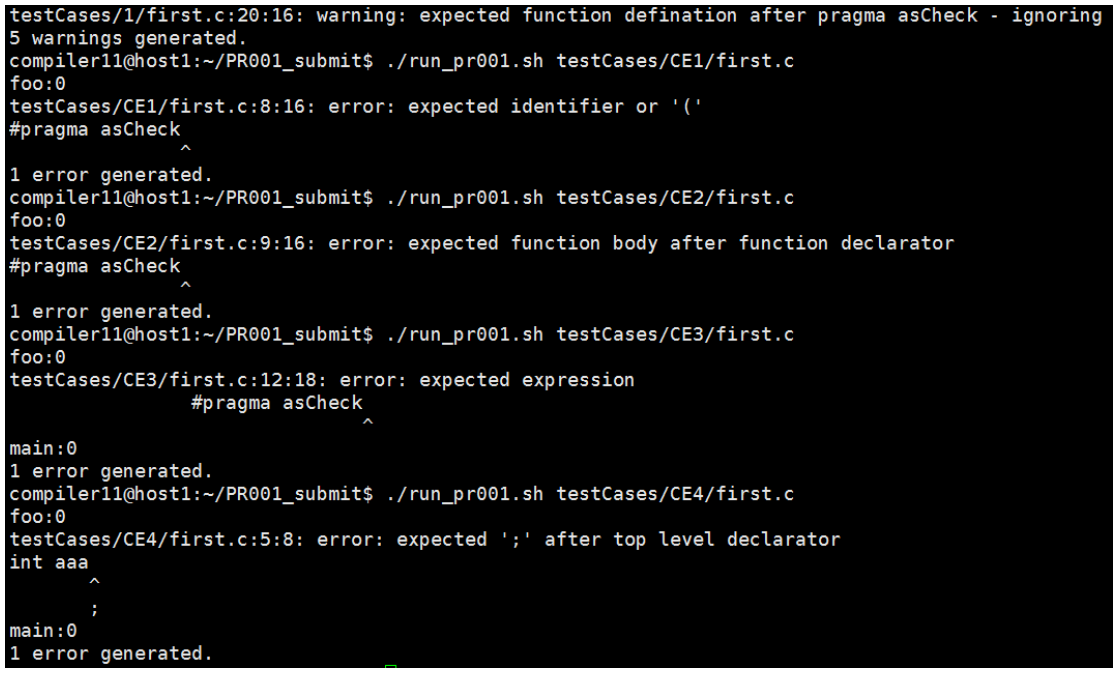
测试截图：



3. 编译错误情况测试

均未导致程序崩溃

测试截图：



4. 运行方式

在~/PR001_submit 目录下运行./run_pr001.sh $path_to_c_file,

其中$path_to_c_file 是指向目标文件的地址