Compiler Construction

Write a Clang plugin

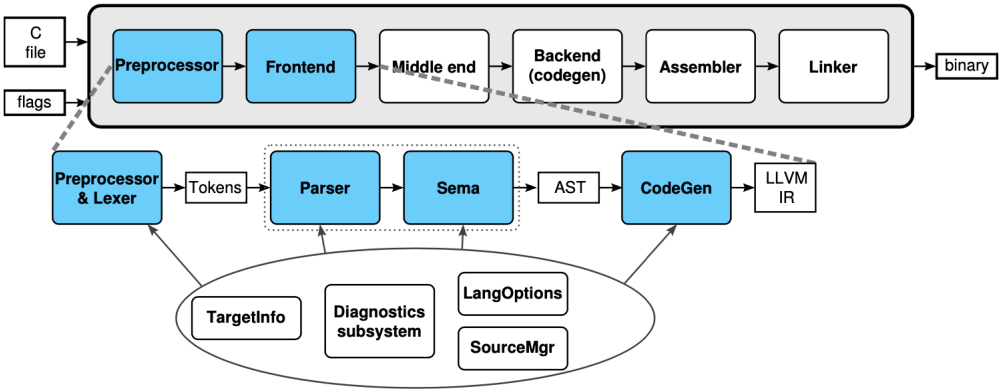
Goal

How to write a Clang plugin?

Objectives:

- Add your own annotations
- Extract some relevant information
- Write an additional optimization pass
- Write DSL
- Generate your own warning
- Writing coverage/documentation tools
- ...

Clang overview



Forestep : install clang (and LLVM)

Clang is big!!

Around than 5,000,000 LOC
+ 200,000,000 LOC llvm-project

Prefer the release mode, the debug mode cost approx 60 Go and several hours of computation.

```
git clone https://github.com/llvm/llvm-project.git
mkdir build
cd build
cmake -DLLVM_ENABLE_PROJECTS=clang -DCMAKE_BUILD_TYPE=Release
```

Clang plugin

- ASTConsumer “consumes” (reads) the AST produced by the Clang parser.
- RecursiveASTVisitor walks recursively the AST

Clang plugin

- PragmaHandler - Used by a plugin to provide new kinds of pragmas.
- ParsedAttrInfo - Used by a plugin to provide new kinds of attributes.
- PluginASTAction - Used by a plugin to provide an AST listener, an object that can observe AST-node creation events.

Which plugin?

```
int func1(int x, int y) {
  return x+y;
}

int func2(int x, int y) {
  return x*y;
}

int saxpy(int a, int x,
          int y) {
  return func1(func2(a,x),y);
}
```

```
int add(int x, int y) {
  return x+y;
}

int multiply(int x, int y) {
  return x*y;
}

int saxpy(int a, int x,
          int y) {
  return add(multiply(a,x),y);
}
```

Writing a plugin (1/9): preliminaries

```
#include "clang/Driver/Options.h"
#include "clang/AST/AST.h"
#include "clang/AST/ASTContext.h"
#include "clang/AST/ASTConsumer.h"
#include "clang/AST/RecursiveASTVisitor.h"
#include "clang/Frontend/ASTConsumers.h"
#include "clang/Frontend/FrontendActions.h"
#include "clang/Frontend/CompilerInstance.h"
#include "clang/Frontend/FrontendPluginRegistry.h"
#include "clang/Rewrite/Core/Rewriter.h"

using namespace std;
using namespace clang;
using namespace llvm;

Rewriter rewriter;
```

Writing a plugin (2/9): define visitor

```
class RenameVisitor : public RecursiveASTVisitor<RenameVisitor> {
private:

    // used for getting additional AST info
    ASTContext *astContext;

public:
    explicit RenameVisitor(CompilerInstance *CI)
        : astContext(&(CI->getASTContext())) // initialize private
    {
        rewriter.setSourceMgr(astContext->getSourceManager(),
            astContext->getLangOpts());
    }
```

Writing a plugin (3/9): define visitor

```
virtual bool VisitFunctionDecl(FunctionDecl *func) {
    string funcName = func->getNameInfo()
                           .getName().getAsString();
    if (funcName == "func1") {
        rewriter.ReplaceText(func->getLocation(),
                             funcName.length(), "add");
    }
    if (funcName == "func2") {
        rewriter.ReplaceText(func->getLocation(),
                             funcName.length(), "multiply");
    }
    return true;
}
```

Writing a plugin (4/9): define consumer

```
    virtual bool VisitStmt(Stmt *st) {
        if (CallExpr *call = dyn_cast<CallExpr>(st)) {
            string callName = call->getDirectCallee()
                                  ->getNameInfo()
                                  .getName().getAsString();
            if(callName == "func1") {
                rewriter.ReplaceText(call->getBeginLoc(),
                                     callName.length(), "add");
            } else if(callName == "func2") {
                rewriter.ReplaceText(call->getBeginLoc(),
                                     callName.length(), "multiply")
            }
        }
        return true;
    }
};
```

Writing a plugin (5/9): arguments and callback

```
class RenameASTConsumer : public ASTConsumer {
private:
    RenameVisitor *visitor; // doesn't have to be private

    // Function to get the base name of the file provided by path
    string basename(std::string path) {
        return std::string( std::find_if(path.rbegin(), path.rend()
    }

    // Used by std::find_if
    struct MatchPathSeparator
    {
        bool operator()(char ch) const {
            return ch == '/';
        }
    };
```

Writing a plugin (6/9): arguments and callback

```
public:
    explicit RenameASTConsumer(CompilerInstance *CI)
        : visitor(new RenameVisitor(CI)) // initialize the visitor
        { }
```

Writing a plugin (3/9): arguments and callback

```
    // This method is called when the ASTs for entire translation
    // unit have been parsed.
    virtual void HandleTranslationUnit(ASTContext &Context) {
        visitor->TraverseDecl(Context.getTranslationUnitDecl());
        // Create an output file to write the updated code
        FileID id = rewriter.getSourceMgr().getMainFileID();
        string filename = "/tmp/" + basename(rewriter.getSourceMgr()
                          .getFilename(rewriter.getSourceMgr().
                          getLocForStartOfFile(id)).str());
        std::error_code OutErrorInfo;
        std::error_code ok;
        llvm::raw_fd_ostream outFile(llvm::StringRef(filename),
            OutErrorInfo, llvm::sys::fs::F_None);
        if (OutErrorInfo == ok) {
            const RewriteBuffer *RewriteBuf = rewriter.getRewriteBufferFor(id);
            outFile << std::string(RewriteBuf->begin(), RewriteBuf->end());
            errs() << "Output file created - " << filename << "\n";
        } else {
            llvm::errs() << "Could not create file\n";
        }
    }
};
```

Writing a plugin (7/9): look for the function

```
class PluginRenameAction : public PluginASTAction {
protected:
    unique_ptr<ASTConsumer> CreateASTConsumer(CompilerInstance &CI,
        return make_unique<RenameASTConsumer>(&CI);
    }

    bool ParseArgs(const CompilerInstance &CI, const vector<string>
        return true;
    }
};

static FrontendPluginRegistry::Add<PluginRenameAction>
    X("-rename-plugin", "simple Plugin example");
```

Writing a plugin (8/9): perform changes

```
add_llvm_library(RenameFunctions MODULE RenameFunctions.cpp
    PLUGIN_TOOL clang)

if(LLVM_ENABLE_PLUGINS AND (WIN32 OR CYGWIN))
  target_link_libraries(RenameFunctions PRIVATE
    clangAST
    clangBasic
    clangFrontend
    LLVMSupport
    )
endif()
```

Test a pluging (9/9)

```
clang -Xclang -load -Xclang RenameFunctions.so
    -Xclang -plugin -Xclang -rename-plugin -c saxpy.c
```

Summary