How to build a program analysis tool using Clang/LLVM 4.0.1

- Initialization of Clang
- Useful functions to print AST
- Line number information of `Stmt`

- Code modification using `Rewriter`
- Converting `Stmt` into String
- Obtaining `SourceLocation`

Initialization of Clang

- Initialization of Clang is complicated
  - To use Clang, many classes should be created and many functions should be called to initialize Clang environment
    - Ex) `ComplierInstance`, `TargetOptions`, `FileManager`, etc.
- It is recommended to use the initialization part of the sample source code from the course homepage *as is*, and implement your own `ASTConsumer` and `RecursiveASTVisitor` classes

Useful functions to print AST

- `dump()` and `dumpColor()` in `Stmt` and `FunctionDecl` to print AST
  - `dump()` shows AST rooted at `Stmt` or `FunctionDecl` object
  - `dumpColor()` is similar to `dump()` but shows AST with syntax highlight
  - Example: dumpColor() of myPrint

```
FunctionDecl 0x368a1e0 <line:6:1> myPrint 'void (int)'
|-ParmVarDecl 0x368a120 <line:3:14, col:18> param 'int'
`-CompoundStmt 0x36a1828 <col:25, line:6:1>
  `-IfStmt 0x36a17f8 <line:4:3, line:5:24>
    |-<<<NULL>>>
    |-BinaryOperator 0x368a2e8 <line:4:7, col:16> 'int' '=='
    | |-ImplicitCastExpr 0x368a2d0 <col:7> 'int' <LValueToRValue>
    | | `-DeclRefExpr 0x368a288 <col:7> 'int' lvalue ParmVar 0x368a120 'param' 'int'
    | `-IntegerLiteral 0x368a2b0 <col:16> 'int' 1
    |-CallExpr 0x368a4e0 <line:5:5, col:24> 'int'
    | |-ImplicitCastExpr 0x368a4c8 <col:5> 'int (*)()' <FunctionToPointerDecay>
    | | `-DeclRefExpr 0x368a400 <col:5> 'int ()' Function 0x368a360 'printf' 'int ()'
    | `-ImplicitCastExpr 0x36a17e0 <col:12> 'char *' <ArrayToPointerDecay>
    |   `-StringLiteral 0x368a468 <col:12> 'char [11]' lvalue "param is 1"
    `-<<<NULL>>>
```

Line number information of `Stmt`

- A `SourceLocation` object from `getLocStart()` of `Stmt` has a line information
  - `SourceManager` is used to get line and column information from `SourceLocation`
    - In the initialization step, `SourceManager` object is created
    - `getExpansionLineNumber()` and `getExpansionColumnNumber()` in `SourceManager` give line and column information, respectively

```
bool VisitStmt(Stmt *s) {
  SourceLocation startLocation = s->getLocStart();
  SourceManager &srcmgr=m_srcmgr;//you can get SourceManager from the initialization part
  unsigned int lineNum = srcmgr.getExpansionLineNumber(startLocation);
  unsigned int colNum = srcmgr.getExpansionColumnNumber(startLocation);
  …
}
```

Code Modification using `Rewriter`

- You can modify code using `Rewriter` class
  - `Rewriter` object can be created with `SourceManager` and `LangOptions` objects which can be obtained from `CompilerInstance` object
    - `CompilerInstance` object provides Clang's initialization configurations
      - see the example in the last page
- Rewriter has functions to insert, remove and replace code
  - InsertTextAfter(*loc,str*), InsertTextBefore(*loc,str*), RemoveText(*loc,size*), ReplaceText(…) , etc. where loc, str, size are a location (SourceLocation), a string, and a size of statement to remove, respectively
- Example: inserting a text before a condition in IfStmt using InsertTextAfter()

```
1 bool MyASTVisitor::VisitStmt(Stmt *s) {
2   if (isa<IfStmt>(s)) {
3     IfStmt *ifStmt = cast<IfStmt>(s);
4     condition = ifStmt->getCond();
5     MyRewriter.InsertTextAfter(condition->getLocStart(), "/*start of cond*/");
6   }
7 }
```

```
if( param == 1 ) ⟶ if( /*start of cond*/param == 1 )
```

Output of Rewriter

- Modified code is obtained from a RewriterBuffer of Rewriter through getRewriteBufferFor()
- Example code which writes modified code in output.txt

```
1 int main(int argc, char *argv[]) {
2   …
3   const RewriteBuffer *RewriteBuf = MyRewriter.getRewriteBufferFor(SourceMgr.getMainFileID());
4   ofstream output("output.txt");
5   output << string(RewriteBuf->begin(), RewriteBuf->end());
6   output.close();
7 }
8
```

Converting Stmt into String

- printPretty(raw_ostream&, PrinterHelper*, PrintingPolicy&) writes a string corresponding to Stmt to raw_ostream
- Example code shows VisitStmt function which gets string from given Stmt
- Check https://stackoverflow.com/a/9639239 for additional information

```
1  bool VisitStmt(Stmt *s) {
2    // MyASTVisitor should receive LangOptions from main as LangOpts
3    clang::PrintingPolicy Policy(LangOpts);
4
5    std::string str1;
6    llvm::raw_string_ostream os(str1);
7    s->printPretty(os, NULL, Policy);
8    llvm::outs() << os.str() << "\n";
9    return true;
10 }
11
```

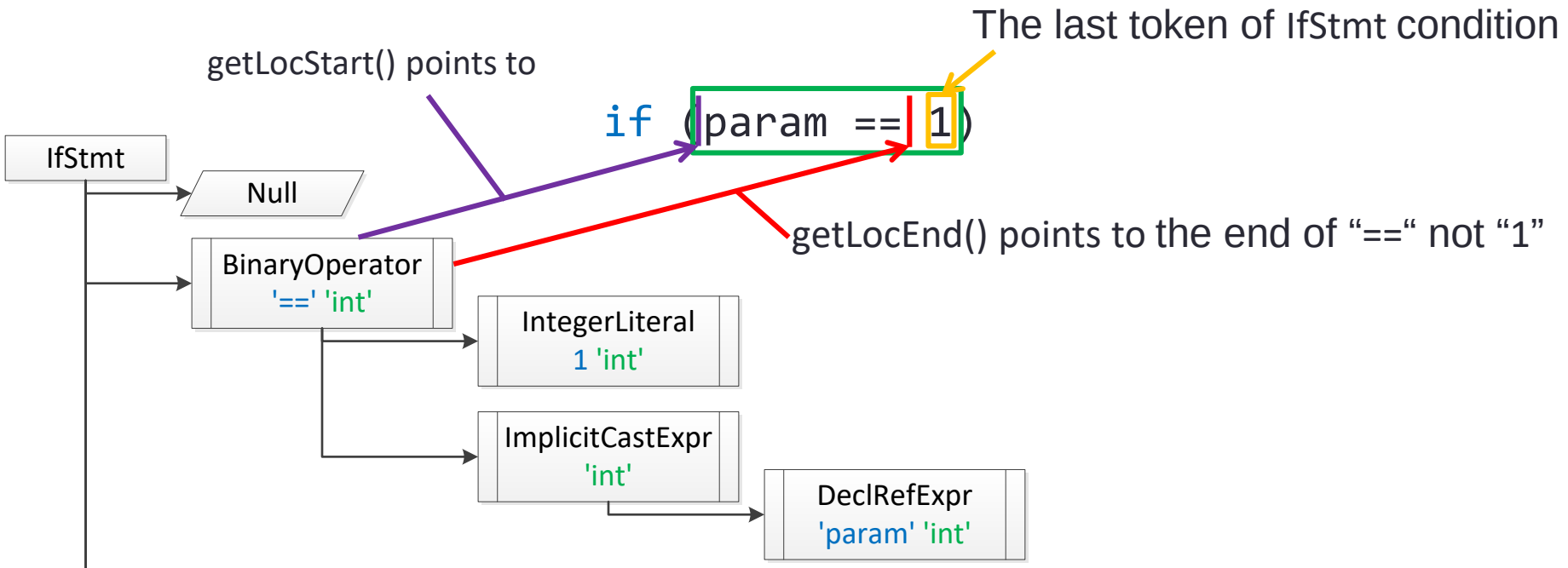
SourceLocation

- To change code, you need to specify where to change
  - Rewriter class requires a SourceLocation class instance which contains location information

- You can get a SourceLocation instance by:
  - getLocStart() and getLocEnd() of Stmt which return a start and an end locations of Stmt instance respectively
  - `findLocationAfterToken`(loc, tok,…) of Lexer which returns the location of the first token tok occurring right after loc
    - Lexer tokenizes a target code
  - SourceLocation.getLocWithOffset(offset,…) which returns location adjusted by the given offset

getLocStart() and getLocEnd()

- getLocStart() returns the exact starting location of Stmt
- getLocEnd() returns the location of Stmt that corresponds to the last-1 th token's ending location of Stmt
  - To get correct end location, you need to use Lexer class in addition
- Example: getLocStart() and getLocEnd() results of IfStmt condition



SourceLocation **getLocWithOffset**(int offset)

```
// Ex.  Logging Caller -> Callee function calls
bool insertProbe(const CallExpr *ce, std::string calleeName) {
   // x = f(a,b); is modified as follows:
   // x = (printf("%s,%s\n",CallerFuncName,calleeName)? f(a,b):0);
   std::string probeFront = "(printf(\"%s,%s\\n\",\""
                +CallerFuncName + "\",\"" + calleeName + "\")?";
   std::string probeBack= ": 0)";

   MyRewriter.InsertTextAfter(ce->getLocStart(), probeFront);
   SourceLocation funEndLoc = ce->getLocEnd().getLocWithOffset(1);
   MyRewriter.InsertTextAfter(funEndLoc, probeBack);

   return true;
}
```

References

- Clang, http://clang.llvm.org/
- Clang API Documentation, http://clang.llvm.org/doxygen/
- How to parse C programs with clang: A tutorial in 9 parts, http://amnoid.de/tmp/clangtut/tut.html

Appendix: Example Source Code (1/5)

- This program prints the name of declared functions, statements and the class name of each Stmt in function bodies

```
1  // PrintFunctions.cpp
2  #include <cstdio>
3  #include <string>
4  #include <iostream>
5  #include <sstream>
6  #include <fstream>
7
8  #include "clang/AST/AST.h"
9  #include "clang/AST/ASTConsumer.h"
10 #include "clang/AST/RecursiveASTVisitor.h"
11 #include "clang/Frontend/ASTConsumers.h"
12 #include "clang/Frontend/CompilerInstance.h"
13 #include "clang/Frontend/FrontendActions.h"
14 #include "clang/Rewrite/Core/Rewriter.h"
15 #include "clang/Tooling/CommonOptionsParser.h"
16 #include "clang/Tooling/Tooling.h"
17 #include "llvm/Support/raw_ostream.h"
18
19 using namespace clang;
20 using namespace clang::driver;
21 using namespace clang::tooling;
22 using namespace std;
23
24 static llvm::cl::OptionCategory MyOptionCategory("MyOptions");
25 static llvm::cl::opt<std::string> OutputFilename("o",
26         llvm::cl::desc("Specify output filename that contains stmt:type"),
27         llvm::cl::value_desc("output_filename"), llvm::cl::cat(MyOptionCategory));
28
29 LangOptions MyLangOpts;
30 SourceManager *ptrMySourceMgr;
31 Rewriter MyRewriter;
```

Appendix: Example Source Code (2/5)

```
33 class MyASTVisitor : public RecursiveASTVisitor<MyASTVisitor> {
34 public:
35     MyASTVisitor() {}
36     bool VisitStmt(Stmt *s) {
37
38         // Print a current statement and its type
39          std::string str1;
40         llvm::raw_string_ostream os(str1);
41         s->printPretty(os, NULL, MyLangOpts);
42
43         puts("-----------------");
44         printf("%s\n", os.str().c_str());
45         printf("TYPE:%s\n", s->getStmtClassName());
46         fflush(stdout);
47         return true;
48     }
49
50     bool VisitFunctionDecl(FunctionDecl *f) { // Print function name
51         llvm::outs() << "**********************************\n";
52         llvm::outs() << "*** FUNCTION NAME:" <<  f->getName() << '\n';
53         llvm::outs() << "**********************************\n";
54         return true;
55     }
56 };
```

Appendix: Example Source Code (3/5)

```
57 class MyASTConsumer : public ASTConsumer {
58 public:
59     MyASTConsumer(): Visitor() {} //initialize MyASTVisitor
60
61     virtual bool HandleTopLevelDecl(DeclGroupRef DR) {
62         for (DeclGroupRef::iterator b = DR.begin(), e = DR.end(); b != e; ++b) {
63             // Travel each function declaration using MyASTVisitor
64             Visitor.TraverseDecl(*b);
65          }
66         return true;
67     }
68
69 private:
70     MyASTVisitor Visitor;
71 };
```

Appendix: Example Source Code (4/5)

```
73 class MyFrontendAction : public ASTFrontendAction {
74 public:
75   MyFrontendAction() {}
76
77   void EndSourceFileAction() override { // Fill out if necessary
78   }
79
80   std::unique_ptr<ASTConsumer> CreateASTConsumer(
81                   CompilerInstance &CI, StringRef file) override {
82
83     MyLangOpts = CI.getLangOpts();
84     ptrMySourceMgr= &(CI.getSourceManager());
85     MyRewriter= Rewriter(*ptrMySourceMgr, MyLangOpts);
86
87     return llvm::make_unique<MyASTConsumer>();
88   }
89 };
```

Appendix: Example Source Code (5/5)

```
 91 int main(int argc, const char **argv) {
 92   CommonOptionsParser op(argc, argv, MyOptionCategory);
 93   ClangTool Tool(op.getCompilations(), op.getSourcePathList());
 94   int rtn_flag;
 95
 96   // ClangTool::run accepts a FrontendActionFactory, which is then used to
 97   // create new objects implementing the FrontendAction interface. Here we use
 98   // the helper newFrontendActionFactory to create a default factory that will
 99   // return a new MyFrontendAction object every time.
100   // To further customize this, we could create our own factory class.
101
102   // AST Parsing
103   rtn_flag= Tool.run(newFrontendActionFactory<MyFrontendAction>().get());
104
105   /* //
106   // Rewriter sample. Save changed target code into output.txt if any
107   const RewriteBuffer *RewriteBuf = MyRewriter.getRewriteBufferFor
108      ((*ptrMySourceMgr).getMainFileID());
109   ofstream out_file ("output.txt");
110   out_file << string(RewriteBuf->begin(), RewriteBuf->end());
111   out_file.close();
112   */
113
114   return rtn_flag;
115 }
```

Appendix: Output on example.c

```
**********************************
*** FUNCTION NAME:myPrint
**********************************
----------------
{
    if (param == 1)
        printf("param is 1");
    for (int i = 0; i < 10; i++) {
        global += i;
    }
}

TYPE:CompoundStmt
----------------
if (param == 1)
    printf("param is 1");

TYPE:IfStmt
----------------
param == 1
TYPE:BinaryOperator
----------------
param
TYPE:ImplicitCastExpr
----------------
param
TYPE:DeclRefExpr
----------------
1
TYPE:IntegerLiteral
----------------
printf("param is 1")
TYPE:CallExpr
----------------
printf
TYPE:ImplicitCastExpr
----------------
printf
TYPE:DeclRefExpr
----------------
"param is 1"
TYPE:ImplicitCastExpr
----------------
"param is 1"
TYPE:ImplicitCastExpr
----------------
"param is 1"
TYPE:StringLiteral
----------------
for (int i = 0; i < 10; i++) {
    global += i;
}

TYPE:ForStmt
----------------
int i = 0;

TYPE:DeclStmt
----------------
0
TYPE:IntegerLiteral
----------------
i < 10
TYPE:BinaryOperator
----------------
i
TYPE:ImplicitCastExpr
----------------
i
TYPE:DeclRefExpr
----------------
10
TYPE:IntegerLiteral
----------------
i++
TYPE:UnaryOperator
----------------
i
TYPE:DeclRefExpr
…
**********************************
*** FUNCTION NAME:main
**********************************
----------------
{
    int param = 1;
    myPrint(param);
    return 0;
}

TYPE:CompoundStmt
----------------
int param = 1;

TYPE:DeclStmt
----------------
1
TYPE:IntegerLiteral
----------------
myPrint(param)
TYPE:CallExpr
----------------
myPrint
TYPE:ImplicitCastExpr
----------------
myPrint
TYPE:DeclRefExpr
----------------
param
TYPE:ImplicitCastExpr
```