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 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     m_rewriter.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
  - ParseAST() modifies a target code as explained in the previous slides
    - TheConsumer contains a Rewriter instance TheRewriter

```
1 int main(int argc, char *argv[]) {
2   …
3   ParseAST(TheCompInst.getPreprocessor(), &TheConsumer, TheCompInst.getASTContext());
4   const RewriteBuffer *RewriteBuf = TheRewriter.getRewriteBufferFor(SourceMgr.getMainFileID());
5   ofstream output(“output.txt”);
6   output << string(RewriteBuf->begin(), RewriteBuf->end());
7   output.close();
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
```

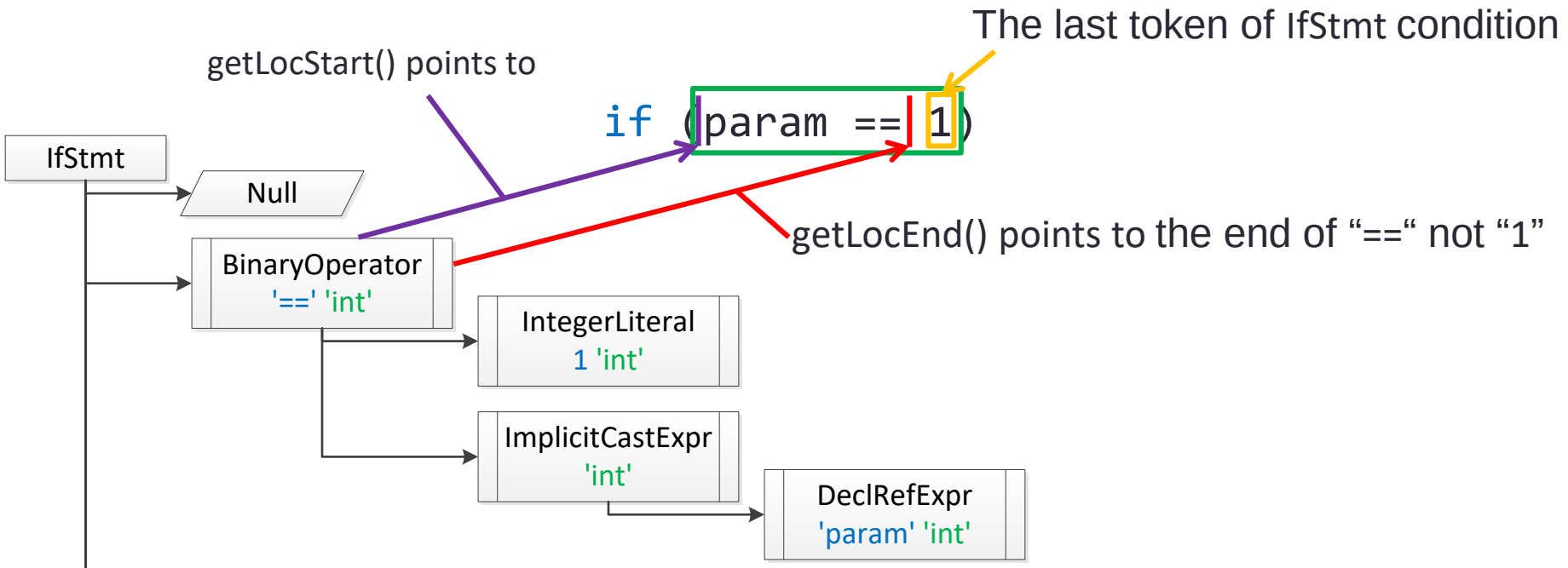
SourceLocation

- 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



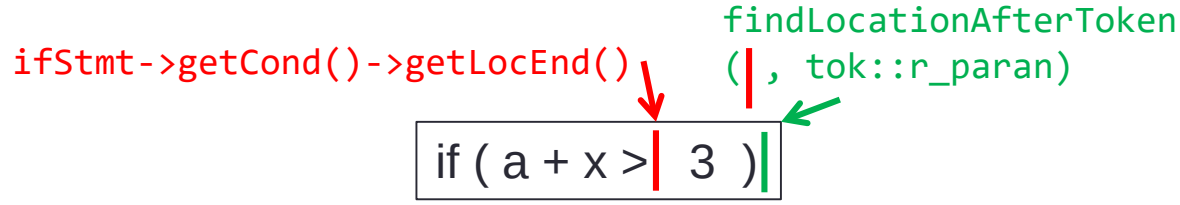
findLocationAfterToken (1/2)

- Static function `findLocationAfterToken(loc,Tkind,…)` of Lexer returns the ending location of the first token of Tkind type after loc

```
static SourceLocation findLocationAfterToken (SourceLocation loc, tok::TokenKind TKind, const
SourceManager &SM, const LangOptions &LangOpts, bool SkipTrailingWhitespaceAndNewLine)
```

- Use `findLocationAfterToken` to get a correct end location of Stmt
  - Example: finding a location of ‘)’ (`tok::r_paren`) using `findLocationAfterToken()` to find the end of if condition

```
1 bool MyASTVisitor::VisitStmt(Stmt *s) {
2   if (isa<IfStmt>(s)) {
3     IfStmt *ifStmt = cast<IfStmt>(s);
4     condition = ifStmt->getCond();
5     SourceLocation endOfCond = clang::Lexer::findLocationAfterToken(condition->
  getLocEnd(), tok::r_paren, m_sourceManager, m_langOptions, false);
6     // endOfCond points ‘)’
7   }
8 }
```



`findLocationAfterToken` (2/2)

- You may find a location of other tokens by changing `TKind` parameter
  - List of useful enums for HW #3

Enum name	Token character
tok::semi	;
tok::r_paren	)
tok::question	?
tok::r_brace	}

- The fourth parameter LangOptions instance is obtained from getLangOpts() of CompilerInstance (see line 99 and line 106 of the appendix)
  - You can find CompilerInstance instance in the initialization part of Clang

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 (1/4)

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

PrintFunctions.c

```
1  #include <cstdio>
2  #include <string>
3  #include <iostream>
4  #include <sstream>
5  #include <map>
6  #include <utility>
7
8  #include "clang/AST/ASTConsumer.h"
9  #include "clang/AST/RecursiveASTVisitor.h"
10 #include "clang/Basic/Diagnostic.h"
11 #include "clang/Basic/FileManager.h"
12 #include "clang/Basic/SourceManager.h"
13 #include "clang/Basic/TargetOptions.h"
14 #include "clang/Basic/TargetInfo.h"
15 #include "clang/Frontend/CompilerInstance.h"
16 #include "clang/Lex/Preprocessor.h"
17 #include "clang/Parse/ParseAST.h"
18 #include "clang/Rewrite/Core/Rewriter.h"
19 #include "clang/Rewrite/Frontend/Rewriters.h"
20 #include "llvm/Support/Host.h"
21 #include "llvm/Support/raw_ostream.h"
22
23 using namespace clang;
24 using namespace std;
25
26 class MyASTVisitor : public RecursiveASTVisitor<MyASTVisitor>
27 {
28 public:
```

Appendix: Example Source Code (2/4)

```
29      bool VisitStmt(Stmt *s) {
30          // Print name of sub-class of s
31          printf("\t%s \n", s->getStmtClassName() );
32          return true;
33      }
34
35      bool VisitFunctionDecl(FunctionDecl *f) {
36                              // Print function name
37          printf("%s\n", f->getName());
38          return true;
39      }
40  };
41
42  class MyASTConsumer : public ASTConsumer
43  {
44  public:
45      MyASTConsumer()
46      : Visitor() //initialize MyASTVisitor
47      {}
48
49      virtual bool HandleTopLevelDecl(DeclGroupRef DR) {
50          for (DeclGroupRef::iterator b = DR.begin(), e = DR.end(); b != e; ++b) {
51              // Travel each function declaration using MyASTVisitor
52              Visitor.TraverseDecl(*b);
53                              }
54          return true;
55      }
56
57  private:
58      MyASTVisitor Visitor;
59  };
60
61
62  int main(int argc, char *argv[])
63  {
```

Appendix: Example Source Code (3/4)

```
64      if (argc != 2) {
65          llvm::errs() << "Usage: PrintFunctions <filename>\n";
66          return 1;
67      }
68
69      // CompilerInstance will hold the instance of the Clang compiler for us,
70      // managing the various objects needed to run the compiler.
71      CompilerInstance TheCompInst;
72
73      // Diagnostics manage problems and issues in compile
74      TheCompInst.createDiagnostics(NULL, false);
75
76      // Set target platform options
77      // Initialize target info with the default triple for our platform.
78      auto TO = std::make_shared<TargetOptions>();
79      TO->Triple = llvm::sys::getDefaultTargetTriple();
80      TargetInfo *TI = TargetInfo::CreateTargetInfo(TheCompInst.getDiagnostics(), TO);
81      TheCompInst.setTarget(TI);
82
83      // FileManager supports for file system lookup, file system caching, and directory search management.
84      TheCompInst.createFileManager();
85      FileManager &FileMgr = TheCompInst.getFileManager();
86
87      // SourceManager handles loading and caching of source files into memory.
88      TheCompInst.createSourceManager(FileMgr);
89      SourceManager &SourceMgr = TheCompInst.getSourceManager();
90
91      // Prreprocessor runs within a single source file
92      TheCompInst.createPreprocessor(TU_Module);
93
94      // ASTContext holds long-lived AST nodes (such as types and decls) .
95      TheCompInst.createASTContext();
96
97      // A Rewriter helps us manage the code rewriting task.
98      Rewriter TheRewriter;
```

Appendix: Example Source Code (4/4)

```
 99      TheRewriter.setSourceMgr(SourceMgr, TheCompInst.getLangOpts());
100
101      // Set the main file handled by the source manager to the input file.
102      const FileEntry *FileIn = FileMgr.getFile(argv[1]);
103      SourceMgr.setMainFileID(SourceMgr.createFileID(FileIn, SourceLocation(), SrcMgr::C_User));
104
105      // Inform Diagnostics that processing of a source file is beginning.
106      TheCompInst.getDiagnosticClient().BeginSourceFile(TheCompInst.getLangOpts(),&TheCompInst.getPreprocessor());
107
108      // Create an AST consumer instance which is going to get called by ParseAST.
109      MyASTConsumer TheConsumer;
110
111      // Parse the file to AST, registering our consumer as the AST consumer.
112      ParseAST(TheCompInst.getPreprocessor(), &TheConsumer, TheCompInst.getASTContext());
113
114      return 0;
115  }
```