A Brief Overview of Clang Generated Abstract Syntax Tree

Gearóid Sheehan R001515123

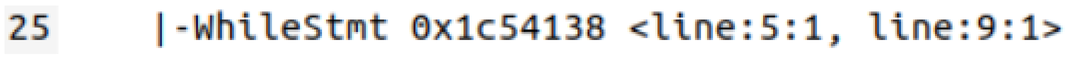
Embedded Systems Engineering COMP8049

The clang compiler is a C language family front end for LLVM and takes care of breaking up C source code into pieces according to a grammatical structure, called an Abstract Syntax Tree. Abbreviated as AST, it is a representation of a programs source code in the form of a tree. The parent child structure of an AST can be seen by the levels of indentation on the left-hand side of the tree.

The t2.ast file is an AST of the t2.c file. This file was generated using the following command:

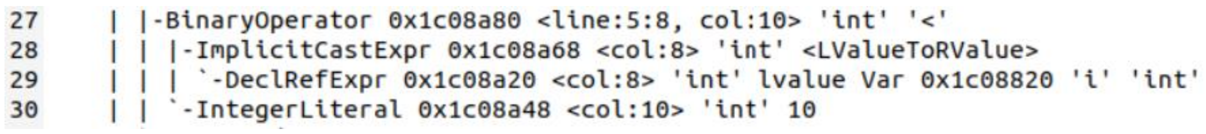
clang-check -ast-dump t2.c --extra-arg="-fno-color-diagnostics" >t2.ast

The section being observed is from line 25 to 53 in t2.ast file. This section includes a WhileStmt and the child nodes within it. This WhileStmt node is declared on line 25, and as with all the other nodes in the AST, is modelled by its data type, address in memory and location by line in the C code.



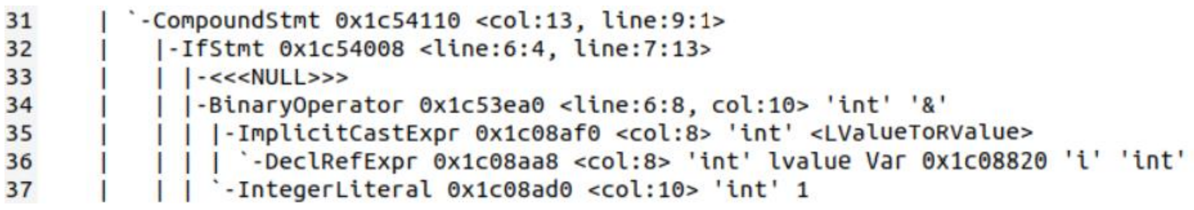
```
25     |-WhileStmt 0x1c54138 <line:5:1, line:9:1>
```

The BinaryOperator node on line 27 is a child node of the WhileStmt and makes a comparison of two integer data types, using the ‘<’ operator. The evaluation of the LValueToRValue node is cast on line 28, and the first child node to be loaded into the BinaryOperator is the integer ‘i’ on line 29. This node is defined in the AST as a DeclRefExpr as it is a reference to the ‘i’ which had already been declared further up the tree. It is then evaluated against the IntegerLiteral ‘10’ on line 30.



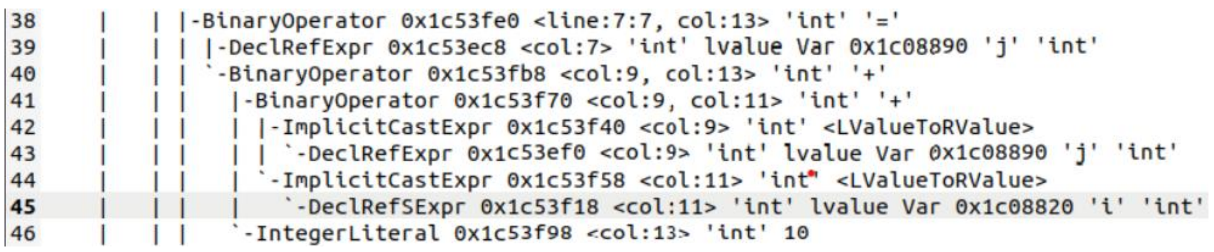
```
27     | |-BinaryOperator 0x1c08a80 <line:5:8, col:10> 'int' '<'
28     | | |-ImplicitCastExpr 0x1c08a68 <col:8> 'int' <LValueToRValue>
29     | | | `-DeclRefExpr 0x1c08a20 <col:8> 'int' lvalue Var 0x1c08820 'i' 'int'
30     | | `-IntegerLiteral 0x1c08a48 <col:10> 'int' 10
```

The CompoundStmt on line 31 is the next child node of the WhileStmt, which allows for multiple statements. It includes two child nodes of its own. The first child node is an IfStmt on line 32, which uses a BinaryOperator just as the WhileStmt before, using the & operand this time. As this IfStmt only has one child node, it does not need a CompoundStmt.



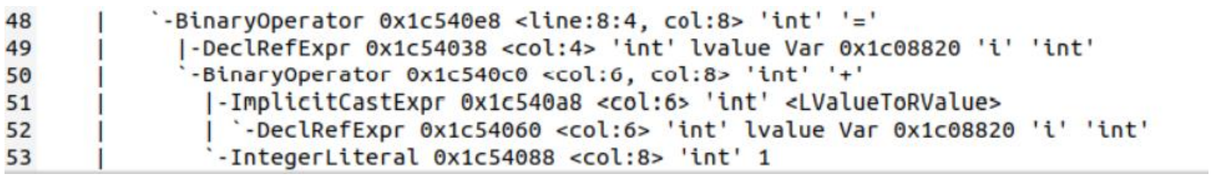
```
31     | `-CompoundStmt 0x1c54110 <col:13, line:9:1>
32     |   |-IfStmt 0x1c54008 <line:6:4, line:7:13>
33     |   | |-<<<NULL>>>
34     |   | |-BinaryOperator 0x1c53ea0 <line:6:8, col:10> 'int' '&'
35     |   | | |-ImplicitCastExpr 0x1c08af0 <col:8> 'int' <LValueToRValue>
36     |   | | | `-DeclRefExpr 0x1c08aa8 <col:8> 'int' lvalue Var 0x1c08820 'i' 'int'
37     |   | | `-IntegerLiteral 0x1c08ad0 <col:10> 'int' 1
```

The expression calculated if the IfStmt is entered is another BinaryOperator on line 38, evaluating the result of the DeclRefExpr ‘j’ when it is added with another DeclRefExpr ‘i’ and the IntegerLiteral ‘10’.



```
38     |   | |-BinaryOperator 0x1c53fe0 <line:7:7, col:13> 'int' '='
39     |   | | |-DeclRefExpr 0x1c53ec8 <col:7> 'int' lvalue Var 0x1c08890 'j' 'int'
40     |   | | `-BinaryOperator 0x1c53fb8 <col:9, col:13> 'int' '+'
41     |   | |   |-BinaryOperator 0x1c53f70 <col:9, col:11> 'int' '+'
42     |   | |   | |-ImplicitCastExpr 0x1c53f40 <col:9> 'int' <LValueToRValue>
43     |   | |   | | `-DeclRefExpr 0x1c53ef0 <col:9> 'int' lvalue Var 0x1c08890 'j' 'int'
44     |   | |   | `-ImplicitCastExpr 0x1c53f58 <col:11> 'int' <LValueToRValue>
45     |   | |   |   `-DeclRefSExpr 0x1c53f18 <col:11> 'int' lvalue Var 0x1c08820 'i' 'int'
46     |   | |   `-IntegerLiteral 0x1c53f98 <col:13> 'int' 10
```

When the IfStmt is exited, a final BinaryOperator is executed at line 48 before the WhileStmt is finished. This expression is almost identical to the last, evaluating the result of the DeclRefExpr ‘i’ when it is added the IntegerLiteral ‘1’.



```
48     |    `-BinaryOperator 0x1c540e8 <line:8:4, col:8> 'int' '='
49     |      |-DeclRefExpr 0x1c54038 <col:4> 'int' lvalue Var 0x1c08820 'i' 'int'
50     |      `-BinaryOperator 0x1c540c0 <col:6, col:8> 'int' '+'
51     |        |-ImplicitCastExpr 0x1c540a8 <col:6> 'int' <LValueToRValue>
52     |        | `-DeclRefExpr 0x1c54060 <col:6> 'int' lvalue Var 0x1c08820 'i' 'int'
53     |        `-IntegerLiteral 0x1c54088 <col:8> 'int' 1
```