# COMS W4115: Programming Assignment 2
Abstract Syntax Tree

## Logistics

1. **Announcement Date:** September $16^{th}$, 2019
2. **Due Date:** October $7^{th}$, 2019 by 5:00pm. No extension!!
3. **Total Points:** 100

## Abstract Syntax Tree

1. Dump AST of bubble.c attached
   (a) Run the following command to generate AST for bubble.c

   ```
   ./build/bin/clang -cc1 -ast-dump bubble.c -I /usr/include
       -I ./build/lib/clang/10.0.0/include
   ```

   Note: In case of "'bits/libc-header-start.h' file not found" error, install the following package.

   ```
   sudo apt-get install gcc-multilib
   ```

   (b) Identify the AST section for bubbleSort function and put them into **{UNI}.txt**.
2. Visualize AST of bubble.c attached
   (a) Run the following command to generate AST in dots file for bubble.c. One dot file will be generated for each function in /tmp.

   ```
   ./build/bin/clang -cc1 -ast-view bubble.c -I /usr/include
       -I ./build/lib/clang/10.0.0/include
   ```

   (b) Identify the AST dot file for bubbleSort function and run the following command to generate graph from the dot file and submit the graph file **{UNI}.pdf**.

   ```
   dot -Tpdf /tmp/AST.dot -o {UNI}.pdf
   ```

3. Generate AST graph for bubbleSort function in more detail

   The AST graph generated in the second task shows only the type of each AST node while the -ast-dump in the first task generates more detail information including values and locations.

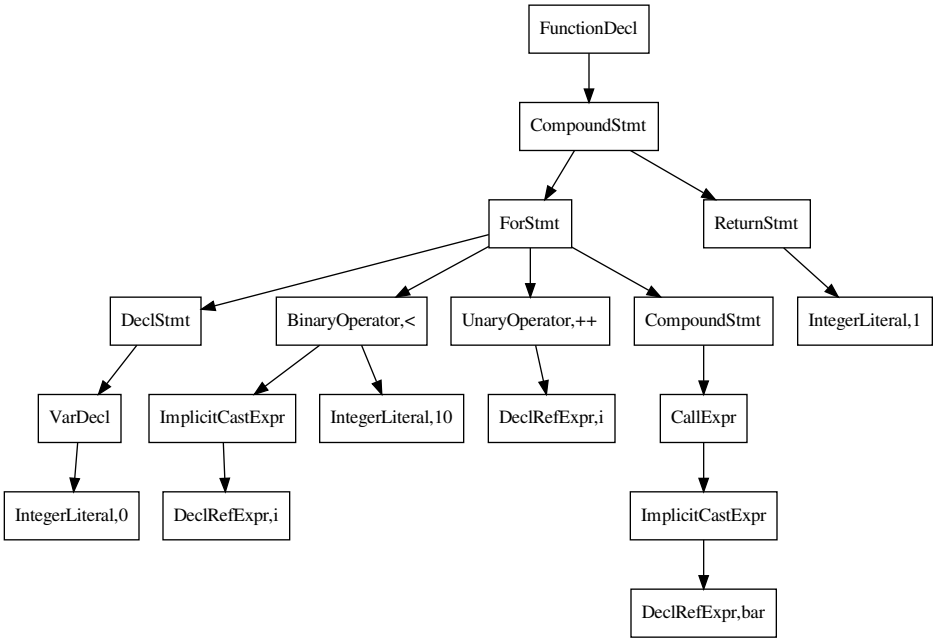
   You should write a Python3 script **{UNI}.py**, which takes **{UNI}.txt** from the first task as input and outputs a dot file. We will run your script as follows,

```
python3 {UNI}.py {UNI}.txt > {UNI}.dot
dot -Tpdf {UNI}.dot -o ast.pdf
```

We will look at the ast.pdf to grade. The graph should at least show detail information for *Operator* nodes, *DeclRefExpr* nodes and *IntegerLiteral* nodes. 10 points for each of the three types. Please don't output location information.

Here is an example graph, which includes more detail information for the following C program.

```
int main(){
    for(int i = 0; i < 10; i++){
        bar();
    }
    return 1;
}
```




4. Write a simple tool based on Clang LibTooling to print the information of all if statements in a C program by using RecursiveASTVisitor.

   (a) Create a new folder "clang-hw2" inside folder "llvm-project/clang/tools/" and put the provided CMakeLists.txt and FindClassDecls.cpp in the new folder. Edit "llvm-project/clang/tools/CMakeLists.txt" and add the following content into the file.

   ```
   add_clang_subdirectory(clang-hw2)
   ```

   Return to the "llvm-project/build" folder and run "make". After it finishes, you can test the example tool by running the following command.

```
./bin/clang-hw2 anycppfile.cpp --
```

It will output the location of all the class declaration in a C++ program. Read the source code and the following Clang website for detail explanation of this example.
https://clang.llvm.org/docs/RAVFrontendAction.html

(b) Now that we know how to create a tool based on Clang LibTooling, you are required to create another Clang LibTooling tool to identify all the *if statements* in a C program and output the information in the following format.

```
If we run:
./bin/clang-hw2 bubble.c --
It should output:
Found IfStmt at 20:7 with Condition: arr[j] > arr[j+1]
```

Rename your C++ file to **{UNI}.cpp** and submit it. We may run your tool on other C programs, all *if statements* should be identified and outputted for full grade.

## Submission Guide

Submit an extra file **contribution.txt** to describe each of your contribution if you work with a partner.

For this assignment, you need to submit four files: **{UNI}.txt**(10 points), **{UNI}.pdf** (10 points), **{UNI}.py**(30 points) and **{UNI}.cpp**(50 points). {UNI} means your UNI number. If you work in pair, {UNI} means UNI1-UNI2.