Assignment 1

Implementation of Nested Functions for C Programs

In this assignment you are required to write a pass in Clang (LLVM's C/C++ compiler frontend) that can do source to source or AST to AST transformations in order to supports nested function (also know as closures) in C language.

Since, C syntax does not support nested functions natively, you will emulate nested function definition using labeled blocks where the label will beconsidered as the name of the nested function. Given nested functions (as labeled blocks), your job will be to do an AST transformations and output a valid AST or valid C code that has the behaviour of nested functions i.e. the block should be hoisted to an outer function and each call site should be updated to pass the required context (all variables captured by the block from enclosing scopes) to this hoisted function. You are forbidden from adding any goto statements in the code during transformation. Nesting should be supported upto arbitrary depth i.e. nested functions within nested functions and so on.

Calls to a nested function should follow static scoping rules. In particular, for variables not declared (as a local variable) inside a nested function, the static scoping rules bind the nearest previous declaration to the same variable name in the definition of nested functions, going from the current function to the outermost one. See below a few examples which illustrate the expected functionality. Note that the examples are not exhaustive!

```
void main(){
  int x=3;
  /*foo is a nested function
   i.e. the below syntax is equivalent to :
   void foo(){
      printf("Value of x inside nested function foo = %d\n",x);
      x=4;
   }
   */

foo:{
  printf("Value of x inside nested function foo = %d\n",x);
  x=4;
  }

  printf("Value of x before calling nested function foo = %d\n",x);/*should
  print 3 after transformation*/

  foo();

  printf("Value of x after calling nested function foo = %d\n",x);/*should print
  4* after transformation/
}
```

Input programs will be required to support only int, float, array, and structure type variables. Your input will be a C program in which all parameters will be assumed to follow call-by-value rules (default of C). After translation, the output will also be a C program or an AST which when compiled produces correct results. Make your assumptions judiciously. Demonstrations of your assignment will be essential. You are allowed to use Clang and LLVM source code only. Instructor's permission will be required to use any other source code or package.

The final assignment submission should be a single tar.gz file containing the following:

1. Full Source code along with a Makefile to build the project. The same should be compilable on any Linux machine with standard libraries installed.
2. A README file containing a very brief description of the various files (in the project) and instructions to build the project.
3. Non-trivial test case programs (at least 5).
4. A detailed report (maximum of 4 pages) describing (i) the design (ii) a high-level description of the changes made in Clang/LLVM to implement the functionality (iii) known bugs/errors in the implementation, (iv) additional features/functionality, if any.

Clarification on the assignment, if any, can be posted in Piazza and will be responded by the TA's as appropriate. You are welcome to discuss with other students. But copying code (as it is or by changing variable names, etc.) will result in the award of zero marks to all parties involved in copying, along with a reduction in final grade. All submissions will be run through plagiarism checkers to find similarities in code.

Do keep a copy of the original LLVM/Clang source with you (i.e. before any modifications/additions from your side).

Deadlines for submission:	11.59 PM on March 10, 2020 .
Weightage:	25%
Evaluation:	Evaluation will be based on the report and a presentation-cum-demo which woud be 20 minutes per student (roughly 5 min. for presentation, 5 min. for Q&A and 10 mts. for demo)