E0255 – Compiler
Assignment -1
Submitted by – Nikita Yadav (15570)

Problem: Implementation of nested functions for C programs.

1. Overview of the solution

I have done source to source transformation using **Libtooling, LibASTMatchers and RecursiveASTVisitor.**

My solution defines global struct variables for each function/labeled block containing pointer to variables defined in that function/labeled block. Apart from adding these struct variables, original source code is modified at two sites which are -
1. Variable declaration
2. Variable references

2 Simple Example

Consider the following source code as given in assignment:

```
#include<stdio.h>
void main()
{
        int x=3;
        /*foo is a nested functioni.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*/
}
```

Following is my proposed solution for above program:

```
#include<stdio.h>
struct main_var {
int* x;
};
struct main_var main_v = { NULL };

void foo ()
{
```

```
        printf("Value of x inside nested function foo = %d\n",(*main_v.x));
        (*main_v.x)=4;
}
void main()
{
        int x=3;
        main_v.x = &x;
        /*foo is a nested functioni.e. the below syntax is equivalent to :
        void 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 print4* after transformation*/
}
```

All the changes are highlighted in red.
I located variable declaration x in main() and added a struct main_var with it's member as a pointer of variable x. This member x is initialised with the address of variable x in main().
Inside foo, wherever x is referenced it's changed to (*main_v.x). In this manner, the changes made in foo get reflected back in main().

3. Solution Details

1. Creation and initialisation of global structs per function/labeled blocks.
- To implement this, **all the variable declarations are found in each function/labeled block and stored in a func_varDecl_map <FuncDecl, set<varDecl>>. Global variables are also stored in this data structure.**
And then global structs are created for each function/labeled block. These structs contain pointers to all the variables declared in the function/labeled block.
- A global struct variable is defined for each global struct, pointer are initialised to NULL.
- Inside the function/labeled block, struct member pointers are assigned address of corresponding variables. This allows the changes made in nested functions reflect back in original/parent functions.

2. Modification of variable references with the help of a hierarchy tree
- **A hierarchy tree is created which tracks the nested function relationship**. For example, if main() has labeled block **foo** and **car**, foo has labeled block **bar** then the hierarchy tree is

```
main    |---> foo |---> bar
        |--->car
```

- For a variable reference in bar, first search variable declarations in bar for a match, if not found search in foo, if not found then main is searched, if not found in main as well then global variables are searched.
- Based on function found in above step, variable reference is modified (using corresponding global struct variable) as shown in above example.

4. Implementation Details

I worked with LLVM 11.0.0 and created this tool using libtooling, AST matchers and recursive AST visitor. C++ source code file clang-nested-func.cpp contain the code for source to source transformation.

5. Functionality Details

Input program support int, float, 1D array and struct type variables. Parameters are also handled properly to support nested functions. My tool can handle recursively nested function blocks, locally and globally defined variables according to static scoping rules, and multiple closures in the same/different functions.

6. Possible problems with this solution

- If a struct already exist with the same name as any of the global struct created per function, this solution will not work. However I have tried to choose safe name.

7. References

1. https://clang.llvm.org/get_started.html
2. https://clang.llvm.org/docs/
3. https://eli.thegreenplace.net/2012/06/08/basic-source-to-source-transformation-with-clang
4. https://eli.thegreenplace.net/2014/05/01/modern-source-to-source-transformation-with-clang-and-libtooling/
5. Some pages from stackoverflow