CSE 504 Homework :

Prateek Roy SBUID : 111481907 NetID: praroy Email: praroy@cs.stonybrook.edu

1) **Analysis of LLVM IR code for loops.**

Code:

```
for(int i = 0; i < 5; ++i)
{
        printf("Hello CSE 504");
}
```

Intermediate Representation (attached file : simple_for.ll):

```
; <label>:7:                                     ; preds = %12, %2
  %8 = load i32, i32* %6, align 4
  %9 = icmp slt i32 %8, 5
  br i1 %9, label %10, label %15

; <label>:10:                                    ; preds = %7
  %11 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([14 x i8], [14 x i8]* @.str, i32 0, i32 0))
  br label %12

; <label>:12:                                    ; preds = %10
  %13 = load i32, i32* %6, align 4
  %14 = add nsw i32 %13, 1
  store i32 %14, i32* %6, align 4
  br label %7

; <label>:15:                                    ; preds = %7
  ret i32 0
}
```

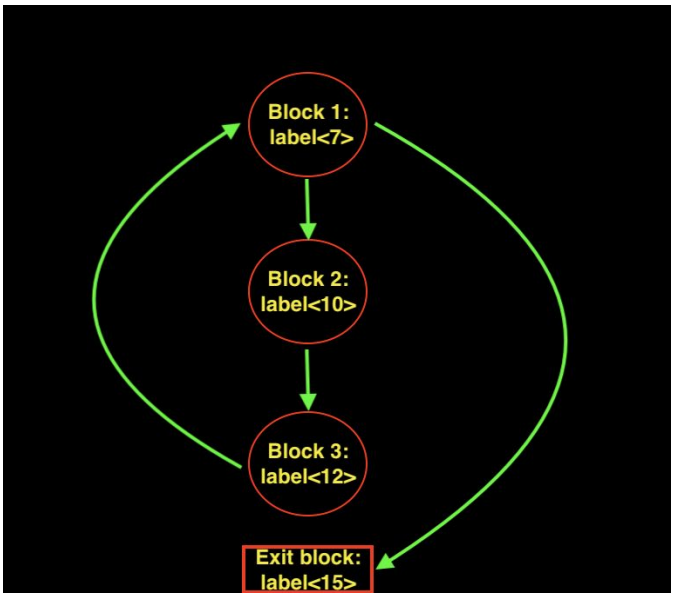
Basic Block 1

Basic Block 2

Basic Block 3

Exit Block

Control Flow Graph :



As you can see from the IR, whenever there is a branch instruction, it acts as a basic block. The for loop is divided into 3 basic blocks and there is a backward edge from block 3 to block 1. This determines there is a loop.

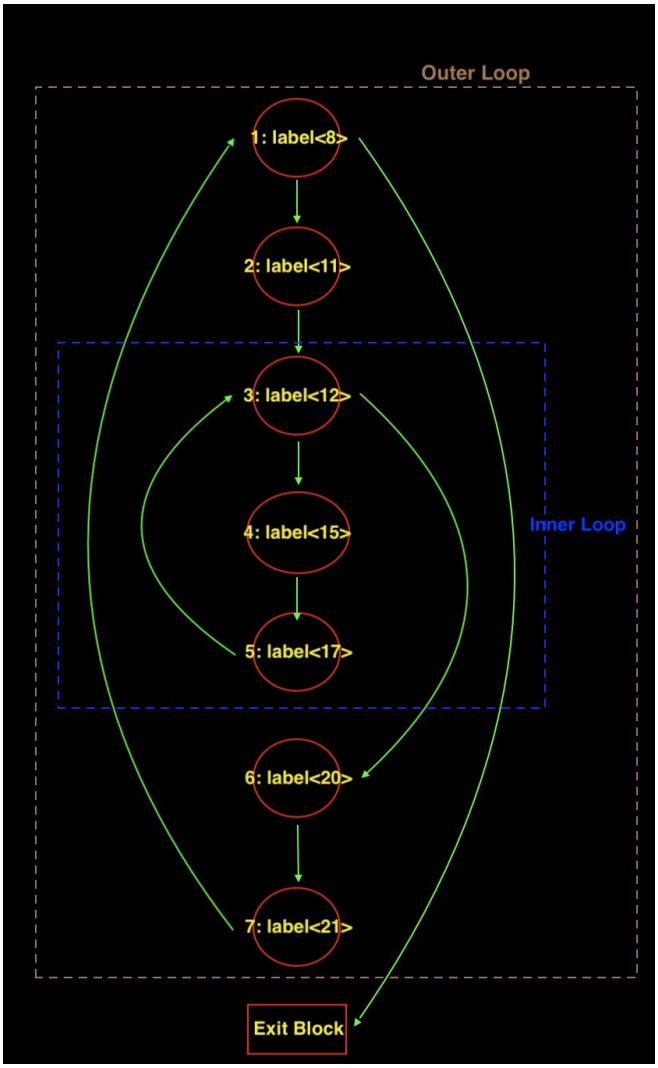
Nested for loop:

Code:

```
for(int i = 0; i < 5; ++i)
{
        for(int j = 0; j < 5; ++j)
                printf("Hello CSE 504");
}
```

The intermediate representation is too big to attach here (attached in zip file : nested_for.ll). But similar to previous explanation, there are 7 branch instructions and thus 7 basic blocks for a nested for loop. The labels are as in the .ll file.

Control Flow Graph:



2) Loop Analysis

I have implemented the pass. Source code : Skeleton.cpp. The design and implementation of the pass is given in next page.

For a simple for loop (code same as code in part 1), mypass prints following output.

```
Loop Info
----------
Loop 0: Block Count : 3 Instructions : 9
```

We can see that a single for loop has 3 basic blocks and 9 instructions in it. This is same as the control flow analysis explained in part 1.

For a nested for loop (code same as code in part 1), mypass prints following output.

```
Loop Info
----------
Loop 0: Block Count : 3 Instructions : 9
Loop 1: Block Count : 7 Instructions : 19
```

We can see that it did identify two loops and the there are 3 blocks in the inner loop and there are 7 basic blocks in the outer loop. Instruction count for inner loop is 9, and for outer loop is 19. This correlates our analysis for control flow graph in part 1.

3) Nested Loop analysis:

To find out nested loop in the Control Flow Graph(CFG), I have saved all the basic blocks which are part of a loop in a data structure called map<int, set<string>>& loopMap, in which the key is loop number and value is the set of basic block numbers . Then in the end, I traverse that data structure to find out whether the basic blocks in a loop is a subset of basic blocks of another loop. In such case, it can be said that the loop whose basic block are subset of basic blocks of another loop is nested within it.
The output for mypass is below for several test cases:

Testcase 1:

```
for(int i = 0; i < 5; ++i)
{
        for(int j = 0; j < 5; ++j)
                printf("Hello CSE 504");
}
```

```
Nested Relationship
-----------------------
Loop 0 is nested within 1
```

Testcase 2:

```
for(int i = 0; i < 5; ++i)
{
        for(int j = 0; j < 5; ++j)
                printf("Hello CSE 504");

        for(int j = 0; j < 5; ++j)
                printf("Hello CSE 504");
}
```

```
Nested Relationship
-----------------------
Loop 1 is nested within 0
Loop 2 is nested within 0
```

4) Sample test file is attached: example_part4.c and IR file: example_part4.ll
Output is :

```
while(0)
{
        printf("Hello");
}

for(int i = 0; i < 5; ++i)
{
        for(int j = 0; j < 5; ++j)
                printf("Hello CSE 504");

        for(int j = 0; j < 5; ++j)
                printf("Hello CSE 504");
}


for(int i = 0; i < 5; ++i)
{
        for(int j = 0; j < 5; ++j)
        {
                for(int k = 0; k < 5; ++k)
                        printf("Hello\n");
        }


        for(int j = 0; j < 5; ++j)
        {
                printf("Hello\n");
        }
}
```

```
Loop Info
---------
Loop 0: Block Count : 3 Instructions : 9
Loop 1: Block Count : 11 Instructions : 30
Loop 2: Block Count : 2 Instructions : 3
Loop 3: Block Count : 3 Instructions : 9
Loop 4: Block Count : 7 Instructions : 19
Loop 5: Block Count : 3 Instructions : 9
Loop 6: Block Count : 15 Instructions : 40
Loop 7: Block Count : 3 Instructions : 9


Nested Relationship
-----------------------
Loop 0 is nested within 1
Loop 3 is nested within 4
Loop 3 is nested within 6
Loop 4 is nested within 6
Loop 5 is nested within 6
Loop 7 is nested within 1
```

Design and implementation of pass:

The pass is implemented in runOnFunction function. According to the lecture slides Page 44 of Slide 13: there are three steps to identify a loop.

Step 1: Construct flowgraph and compute the **dominance relation** for it.
construct_flowgraph() function does this step. **Flowgraph** structure is used to define a graph which internally has following data structures:
blocks data structure which stores all the basic blocks in the IR represented by block name.
preds is used to store all the predecessor blocks of a block.
edges store all the edge between blocks in the IR.

Dominance Tree(dominTree) is a important data structure which stores the dominance relationship between nodes. So for each block, we store all the blocks who dominates it. n dominates m iff every path from s to m contains n. We achieve this in dom_comp() function where the dominators of a block is updated by all common dominators of its predecessors. This update goes on until all the blocks get relaxed in terms of updating dominators i.e there is no change in dominator list of blocks.

Step 2: Find backward edges.
find_backwardedges() function does this. So basically edges are searched who have a backward path in dominance tree. A backward edge is one whose direction is inverse to the direction of dominance. So it goes from a node to a dominator of that node. And this backward edge is the starting/entry point of a loop. For example if block 1 dominates block 2, but there is an edge from 2 to 1, then it is considered as a backward edge. These are stored in **backwardedges** data structure.

Step 3: Construct the loop by backward edges and predecessors by traversing the flowgraph in a reverse order by using predecessor data structure. We know the backward edge is the starting point of a loop, then we do a depth first search on the vertices of those edge using the predecessor nodes in backward order.
Loops data structure is used to store the basic blocks which are part of a loop.
LoopMap stores the loop data(basic blocks) corresponding to loop id.
BlockMap stores the basic block pointer corresponding to a blockid.
For calculating **nested loops**, we search a loop data (basic blocks associated with a loop) across all the loops to find whether it is a subset of any other loop. If we find a match, then there is a nested relationship.

Code of the pass : Skeleton.cpp
Run the pass :
opt -load ~/llvm-pass-skeleton/build/skeleton/libSkeletonPass.* -mypass example.ll