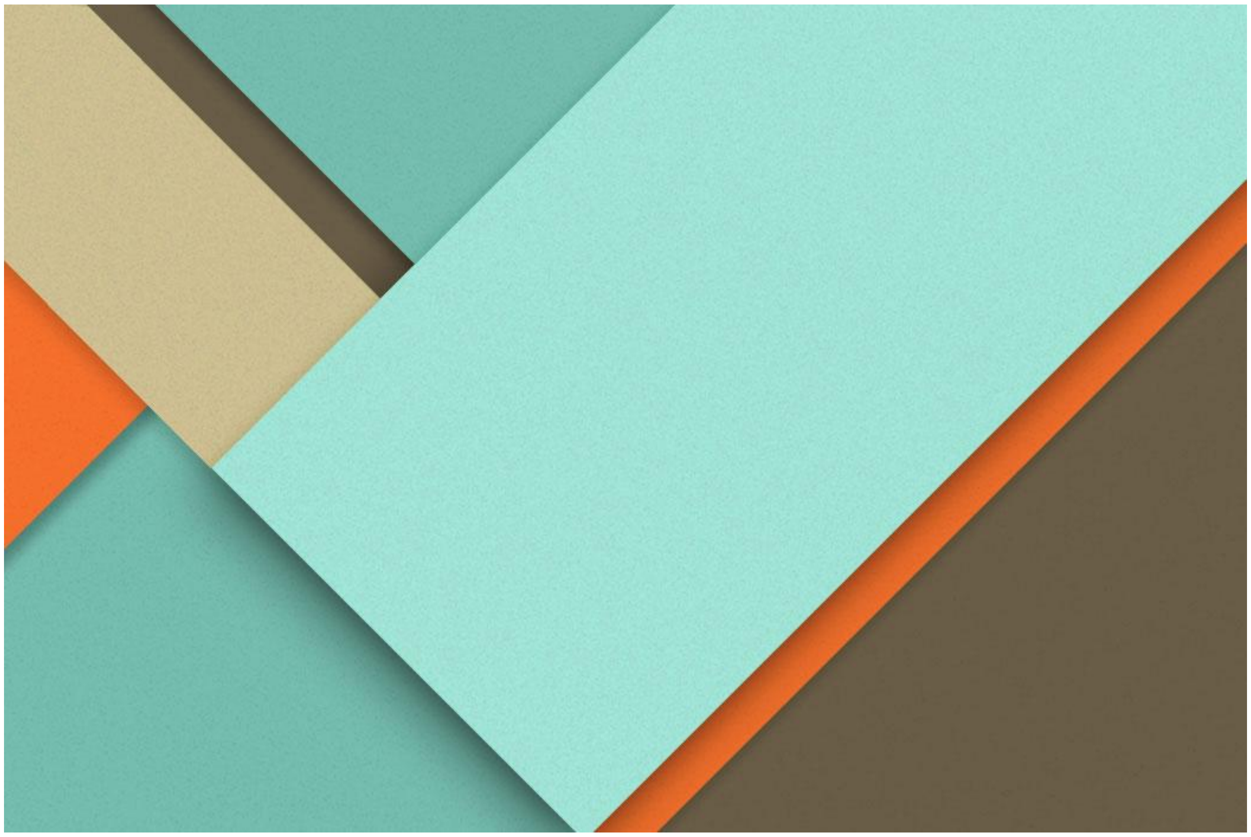


LP ASSIGNMENT

11.04.2021

TEJOMAY PADOLE
BT18CSE034

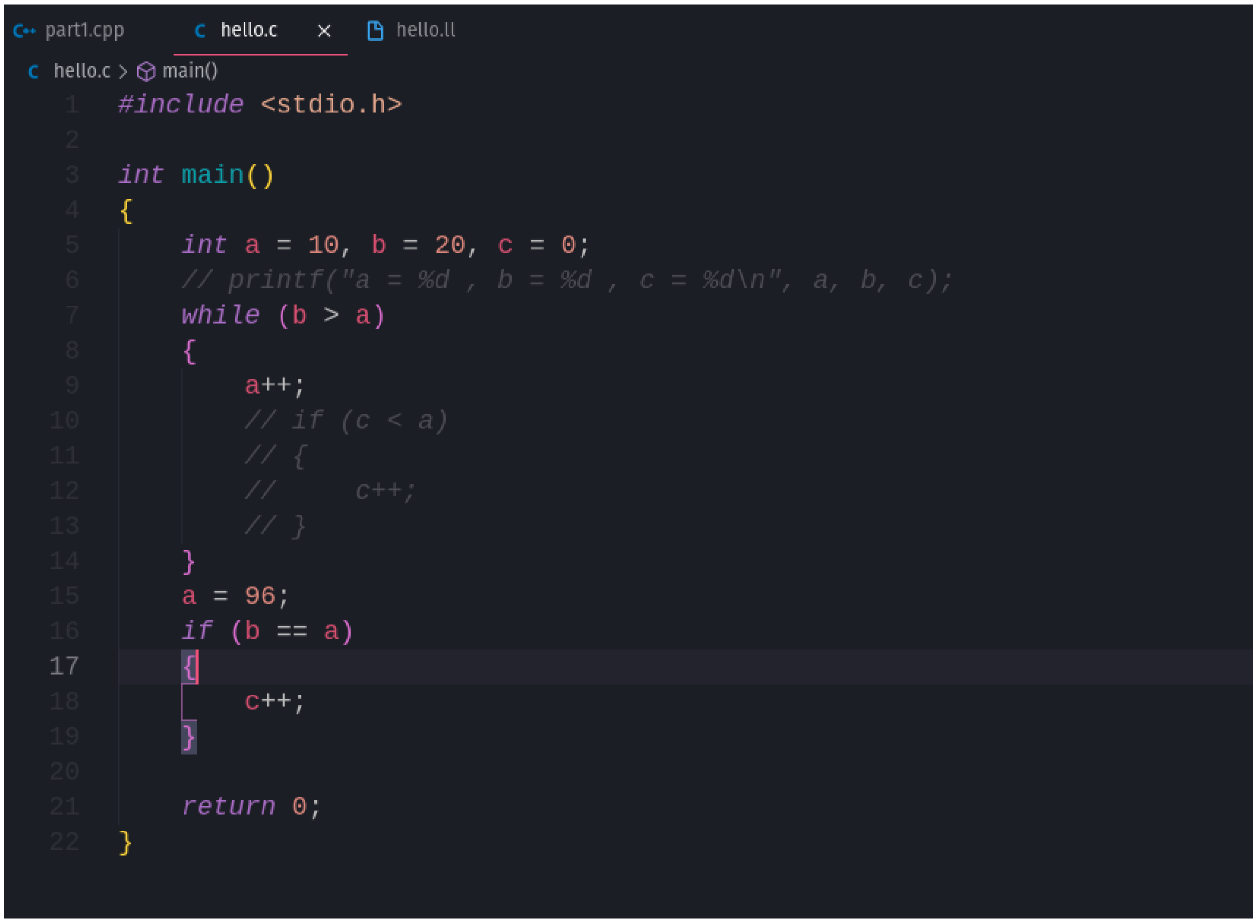
ATHARVA KULKARNI
BT18CSE074

Overview

We created basic blocks from code that is in 3-address form and performed local optimizations. For given code, our program creates a control flow graph. Using the graph, we find out if there is any unreachable code in the sample code. Also, the program points out natural loops and dominator blocks.

(Make sure that you have LLVM and Clang installed in your system to test the given code)

Program input



```c
#include <stdio.h>

int main()
{
    int a = 10, b = 20, c = 0;
    // printf("a = %d , b = %d , c = %d\n", a, b, c);
    while (b > a)
    {
        a++;
        // if (c < a)
        // {
        //     c++;
        // }
    }
    a = 96;
    if (b == a)
    {
        c++;
    }

    return 0;
}
```

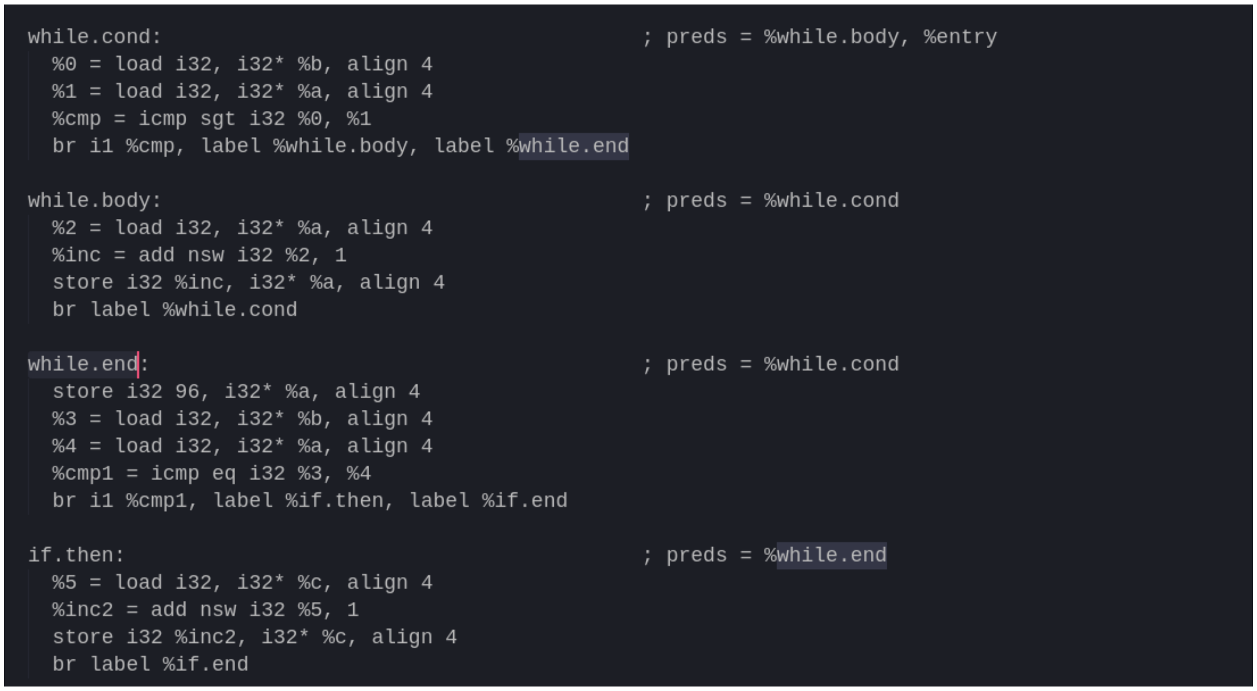
----

1. The above snippet shows the input C file for which 3 address code will be generated through clang.

2. This is the 3-address code generated for the given C code.

   This will be used as Intermediate Representation (IR) for actual code.

   (The IR for LLVM is strongly typed)



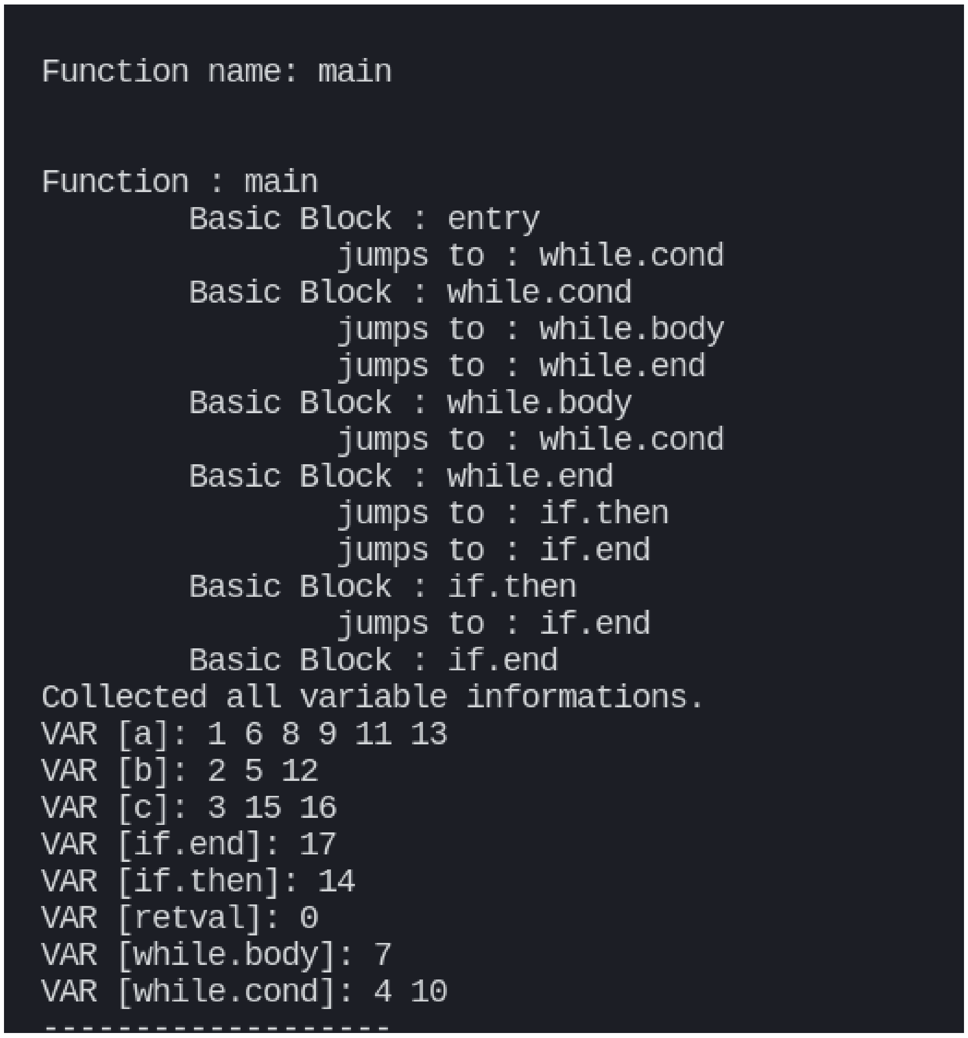
```
while.cond:                                       ; preds = %while.body, %entry
  %0 = load i32, i32* %b, align 4
  %1 = load i32, i32* %a, align 4
  %cmp = icmp sgt i32 %0, %1
  br i1 %cmp, label %while.body, label %while.end

while.body:                                       ; preds = %while.cond
  %2 = load i32, i32* %a, align 4
  %inc = add nsw i32 %2, 1
  store i32 %inc, i32* %a, align 4
  br label %while.cond

while.end:                                        ; preds = %while.cond
  store i32 96, i32* %a, align 4
  %3 = load i32, i32* %b, align 4
  %4 = load i32, i32* %a, align 4
  %cmp1 = icmp eq i32 %3, %4
  br i1 %cmp1, label %if.then, label %if.end

if.then:                                          ; preds = %while.end
  %5 = load i32, i32* %c, align 4
  %inc2 = add nsw i32 %5, 1
  store i32 %inc2, i32* %c, align 4
  br label %if.end
```

Program output

Control flow graph

Below output shows a simplified control flow graph, it displays information regarding the connections of basic blocks. This graph would then help us find unreachable code and dominator blocks.

Following Image also shows which definitions (number) are being generated for which variables.



```
Function name: main


Function : main
        Basic Block : entry
                jumps to : while.cond
        Basic Block : while.cond
                jumps to : while.body
                jumps to : while.end
        Basic Block : while.body
                jumps to : while.cond
        Basic Block : while.end
                jumps to : if.then
                jumps to : if.end
        Basic Block : if.then
                jumps to : if.end
        Basic Block : if.end
Collected all variable informations.
VAR [a]: 1 6 8 9 11 13
VAR [b]: 2 5 12
VAR [c]: 3 15 16
VAR [if.end]: 17
VAR [if.then]: 14
VAR [retval]: 0
VAR [while.body]: 7
VAR [while.cond]: 4 10
-------------------
```

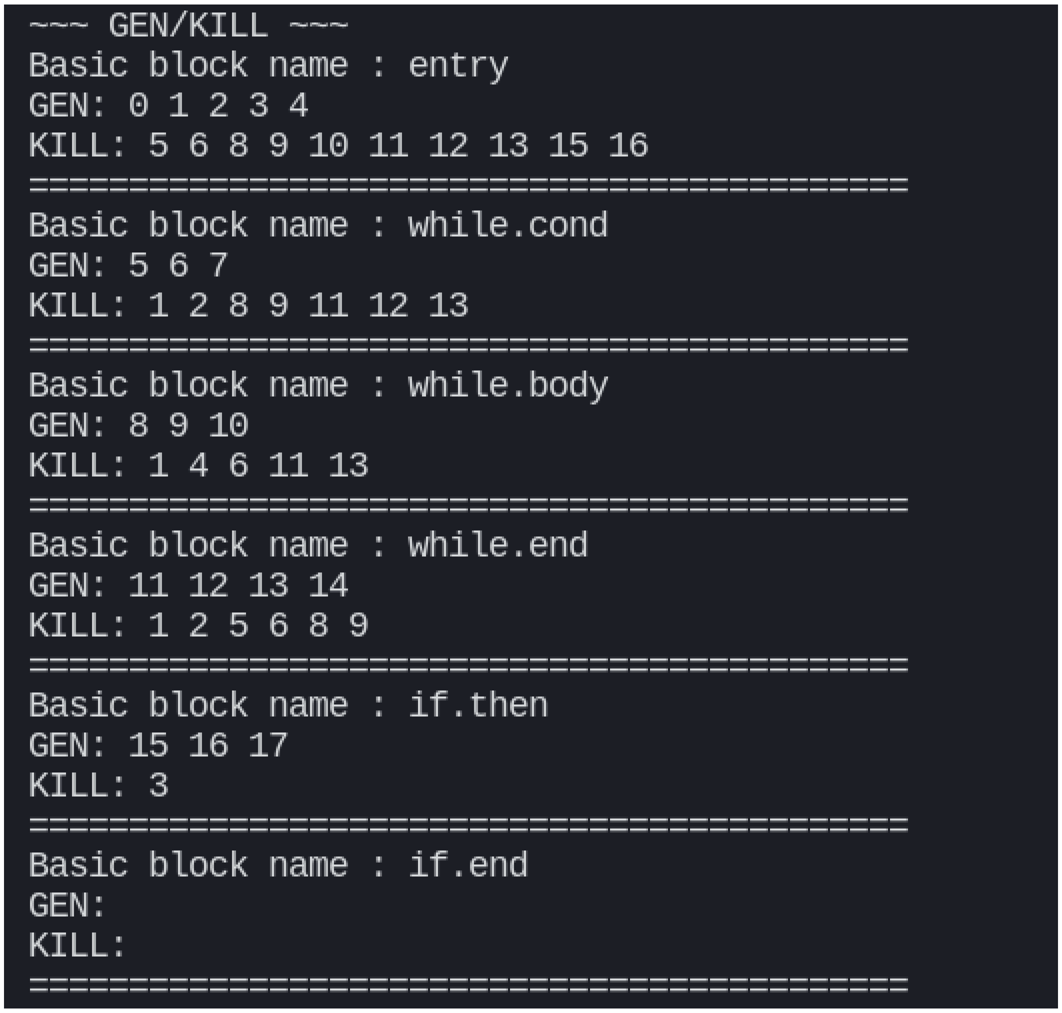
GEN/ KILL

The below output displays the GEN and KILL for every basic block created for our sample code.

Referring to the previous output, the GEN/KILL blocks can be explained as follows:

GEN: 0 1 2 3 4 , the numbers refer to the variables in particular block. In the previous code, we can see that 0 corresponds to variable “retval”, 1 corresponds to variable “a” and so on.

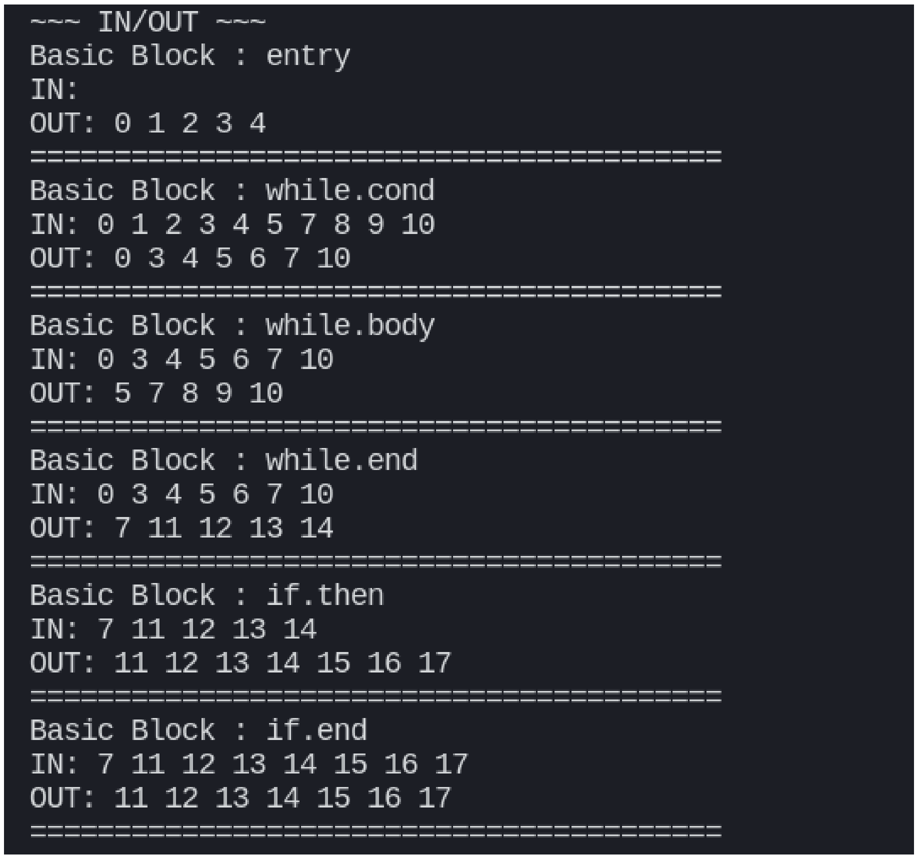
So, for basic block entry, GEN variables are retval, a, b, c and while.cond.



```
~~~ GEN/KILL ~~~
Basic block name : entry
GEN: 0 1 2 3 4
KILL: 5 6 8 9 10 11 12 13 15 16
============================================
Basic block name : while.cond
GEN: 5 6 7
KILL: 1 2 8 9 11 12 13
============================================
Basic block name : while.body
GEN: 8 9 10
KILL: 1 4 6 11 13
============================================
Basic block name : while.end
GEN: 11 12 13 14
KILL: 1 2 5 6 8 9
============================================
Basic block name : if.then
GEN: 15 16 17
KILL: 3
============================================
Basic block name : if.end
GEN:
KILL:
============================================
```

IN/ Out:

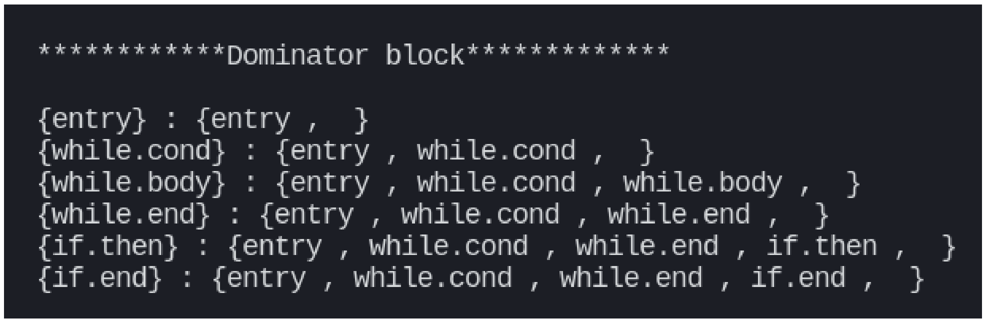
Following image shows the calculated in and out definitions for the given block. Similar to how GEN/KILL are displayed, IN/OUT values also show the same correspondence to variables.



```
~~~ IN/OUT ~~~
Basic Block : entry
IN:
OUT: 0 1 2 3 4
=========================================
Basic Block : while.cond
IN: 0 1 2 3 4 5 7 8 9 10
OUT: 0 3 4 5 6 7 10
=========================================
Basic Block : while.body
IN: 0 3 4 5 6 7 10
OUT: 5 7 8 9 10
=========================================
Basic Block : while.end
IN: 0 3 4 5 6 7 10
OUT: 7 11 12 13 14
=========================================
Basic Block : if.then
IN: 7 11 12 13 14
OUT: 11 12 13 14 15 16 17
=========================================
Basic Block : if.end
IN: 7 11 12 13 14 15 16 17
OUT: 11 12 13 14 15 16 17
=========================================
```

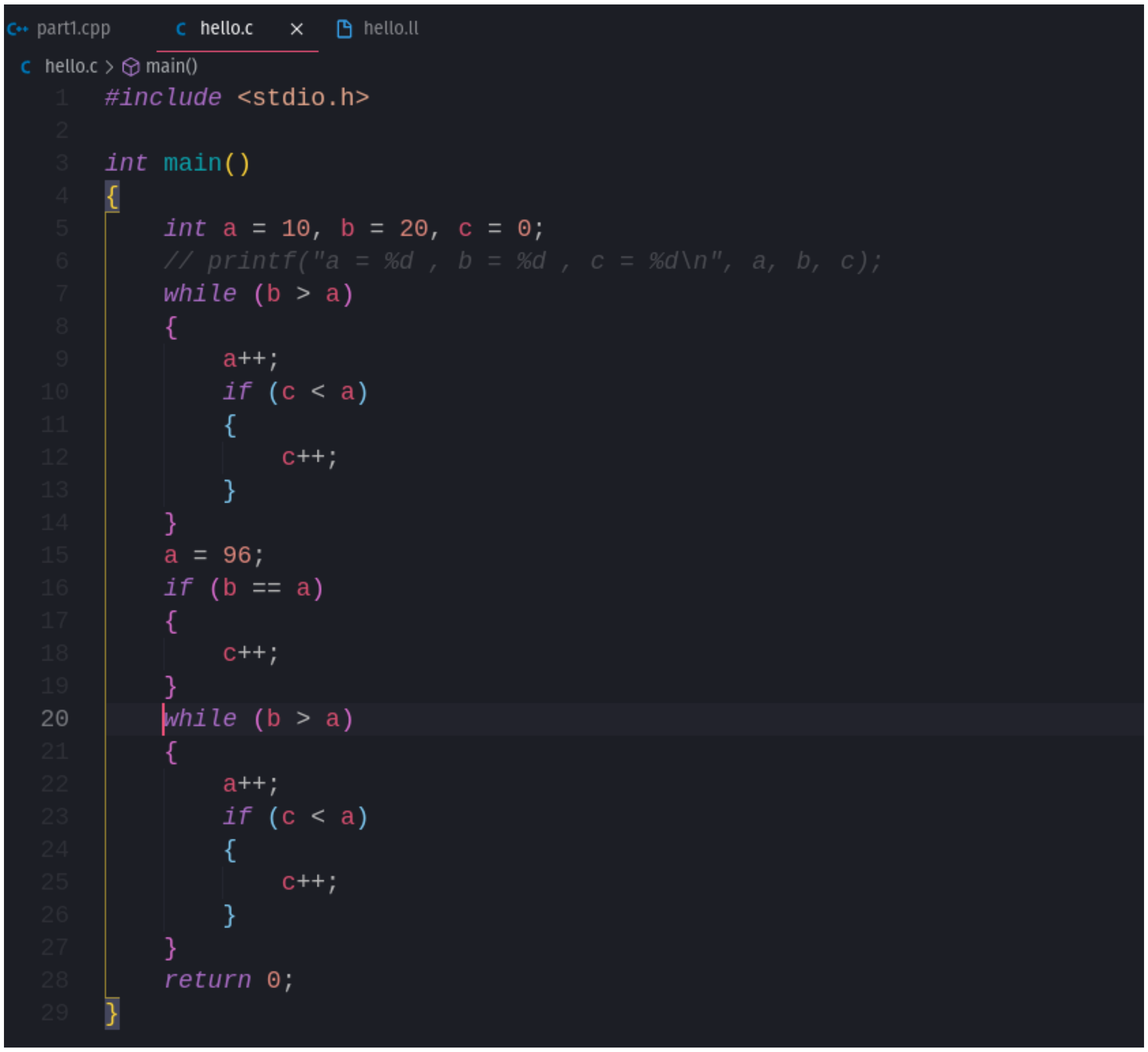
Dominator Block:

Following Image shows which basic blocks dominate the given basic block. This is for the above displayed sample code.



```
************Dominator block*************

{entry} : {entry ,  }
{while.cond} : {entry , while.cond ,  }
{while.body} : {entry , while.cond , while.body ,  }
{while.end} : {entry , while.cond , while.end ,  }
{if.then} : {entry , while.cond , while.end , if.then ,  }
{if.end} : {entry , while.cond , while.end , if.end ,  }
```

A complex example for natural loops and dominator blocks:

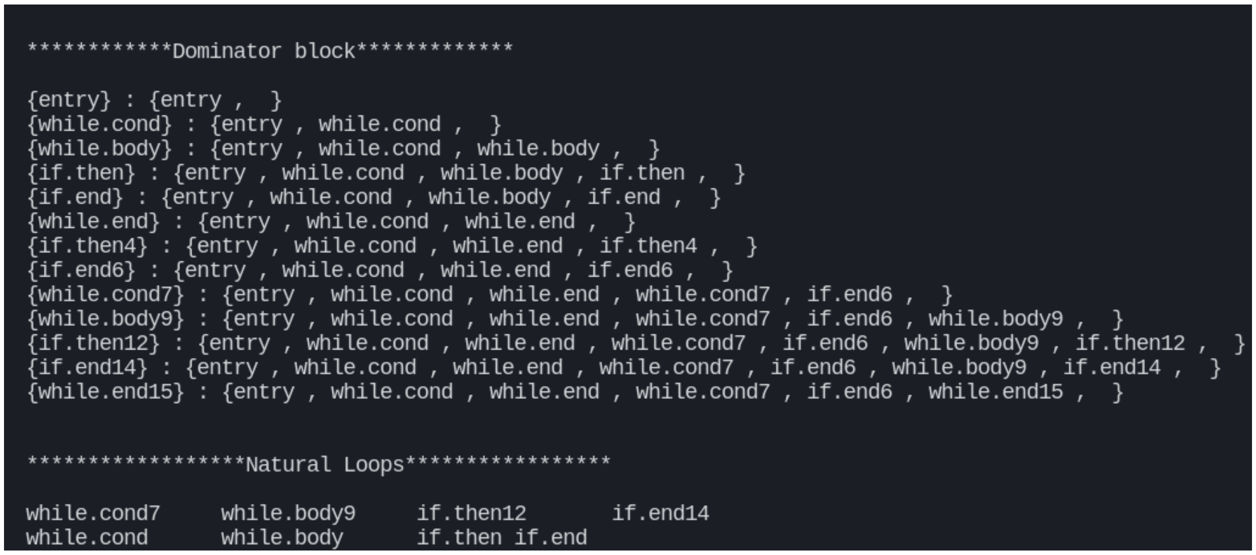


```c
#include <stdio.h>

int main()
{
    int a = 10, b = 20, c = 0;
    // printf("a = %d , b = %d , c = %d\n", a, b, c);
    while (b > a)
    {
        a++;
        if (c < a)
        {
            c++;
        }
    }
    a = 96;
    if (b == a)
    {
        c++;
    }
    while (b > a)
    {
        a++;
        if (c < a)
        {
            c++;
        }
    }
    return 0;
}
```

Above code was used in order to show dominator blocks and natural loops in a better way.

Natural loops and Dominator blocks:

The below output displays dominator blocks and natural loops for the above code. As code complexity increased, the number of dominator blocks and natural loops increased as well, program displays all of them one by one.



```
************Dominator block*************

{entry} : {entry ,  }
{while.cond} : {entry , while.cond ,  }
{while.body} : {entry , while.cond , while.body ,  }
{if.then} : {entry , while.cond , while.body , if.then ,  }
{if.end} : {entry , while.cond , while.body , if.end ,  }
{while.end} : {entry , while.cond , while.end ,  }
{if.then4} : {entry , while.cond , while.end , if.then4 ,  }
{if.end6} : {entry , while.cond , while.end , if.end6 ,  }
{while.cond7} : {entry , while.cond , while.end , while.cond7 , if.end6 ,  }
{while.body9} : {entry , while.cond , while.end , while.cond7 , if.end6 , while.body9 ,  }
{if.then12} : {entry , while.cond , while.end , while.cond7 , if.end6 , while.body9 , if.then12 ,  }
{if.end14} : {entry , while.cond , while.end , while.cond7 , if.end6 , while.body9 , if.end14 ,  }
{while.end15} : {entry , while.cond , while.end , while.cond7 , if.end6 , while.end15 ,  }


******************Natural Loops*****************

while.cond7     while.body9     if.then12       if.end14
while.cond      while.body      if.then if.end
```