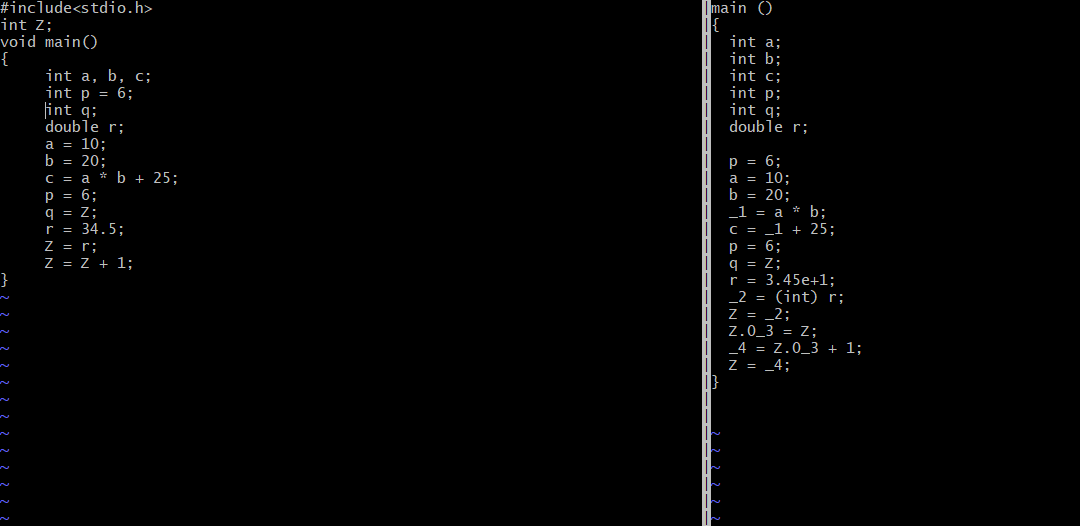
PPL ASSIGNMENT1

MIS: 111903088, DIV :2

NAME: SANSKAR SANJAY DUDHE

A-1 Declarations, Expressions, and Assignments

A-1 SOLUTIONS:



SOLUTIONS:

1 The declaration are shown independently on its own line as a

declaration of local variables. Global variables have no such type

of declarations and are used directly. Set of Initializers are converted

into simple declarations at the top and assignment statement in the

body of code.

2 The statements in gimple can have at most 1 operations. If an expressi-

Ion contains more than one operation then it is broken by introducing

Temporary variables .C expressions are converted to Gimple statements

By abstract syntax trees wherein each non-leaf node specifies either a

Local or temporary variables storage.

3 Floats/doubles are represented as scientific notation in gimple

When Float/Doubles are assigned to an integer,then they are

Converted to explicit cases in Gimple.

4 Reading/writing from/to global variables requires a memory load

or store operation. As GIMPLE statements can have at most one

operation, the only type of statements involving globals are of the

form "GLOBAL = LOCAL" or "LOCAL = GLOBAL". This is because while

local variables of a procedure can reside entirely in registers,

every read/write to global variables must refer to main memory as

they can be accessed by other entities (threads, I/O devices, etc).

Hence, any operation on global variables (such as the increment

shown in this example) has to go through an indirection of a

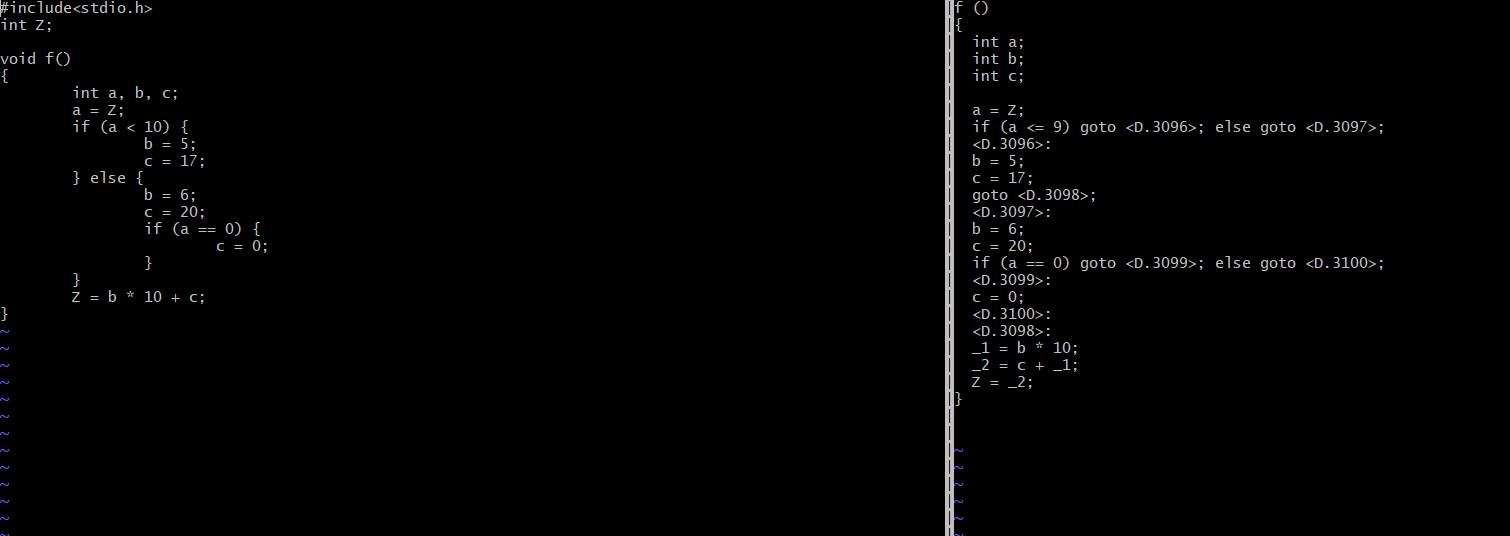
temporary version of the global inside registers (as this temporary

value is not visible to I/O devices, etc). Global temporaries are

named similar to the global variable but with a numeric suffix.

A-2 SOLUTIONS

(CONDITIONAL JUMPS AND CONTROL FLOW GRAPHS)



1 If-blocks have been broken down into distinct sections of code with

labels (named similar to temporary variables). Control transfer

occurs with unconditional jumps (e.g. "goto X") or conditional jumps

(e.g. "if Z then goto X else goto Y").

2 The control-flow graph dump shows distinct basic blocks which are

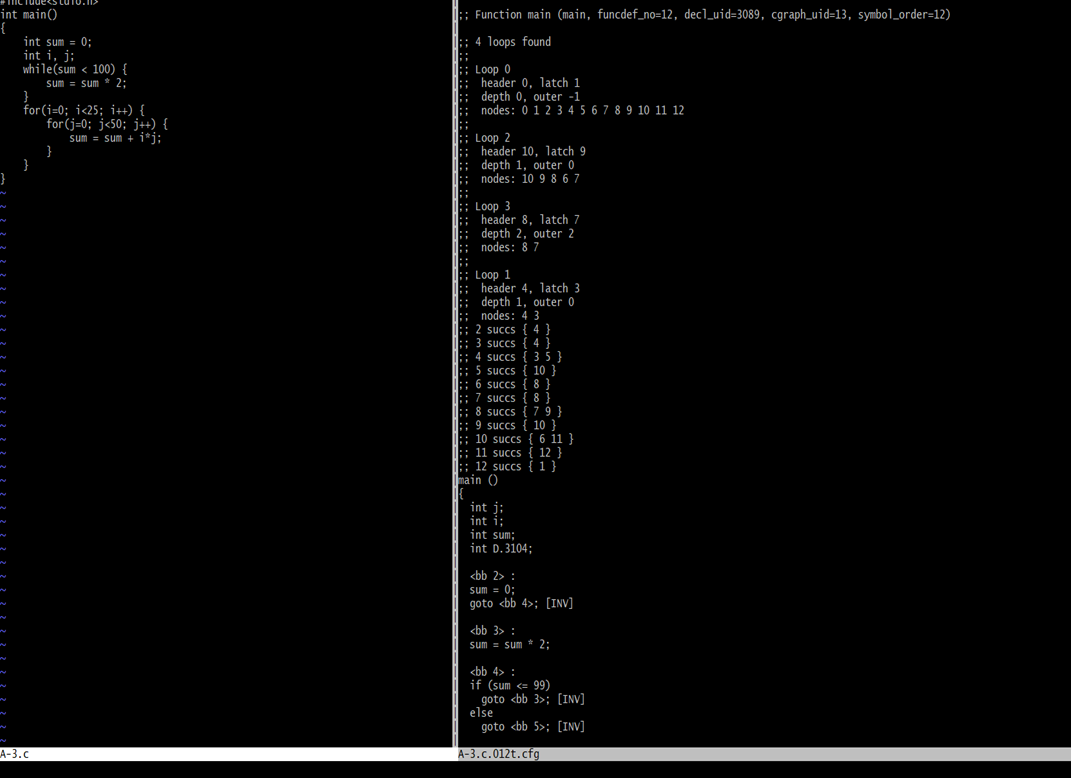
named sequentially starting from 1. Basic blocks roughly correspond

to labels that are created during the gimplification of control

statements.

A-3 SOLUTIONS:

(LOOPS)



1 Basic blocks which are named as2,3,4 are while loops which is set of

Blocks with conditional control the difference being a jump to block

that has been previously executed.

2 A for-loop is a while-loop with the initializer having an

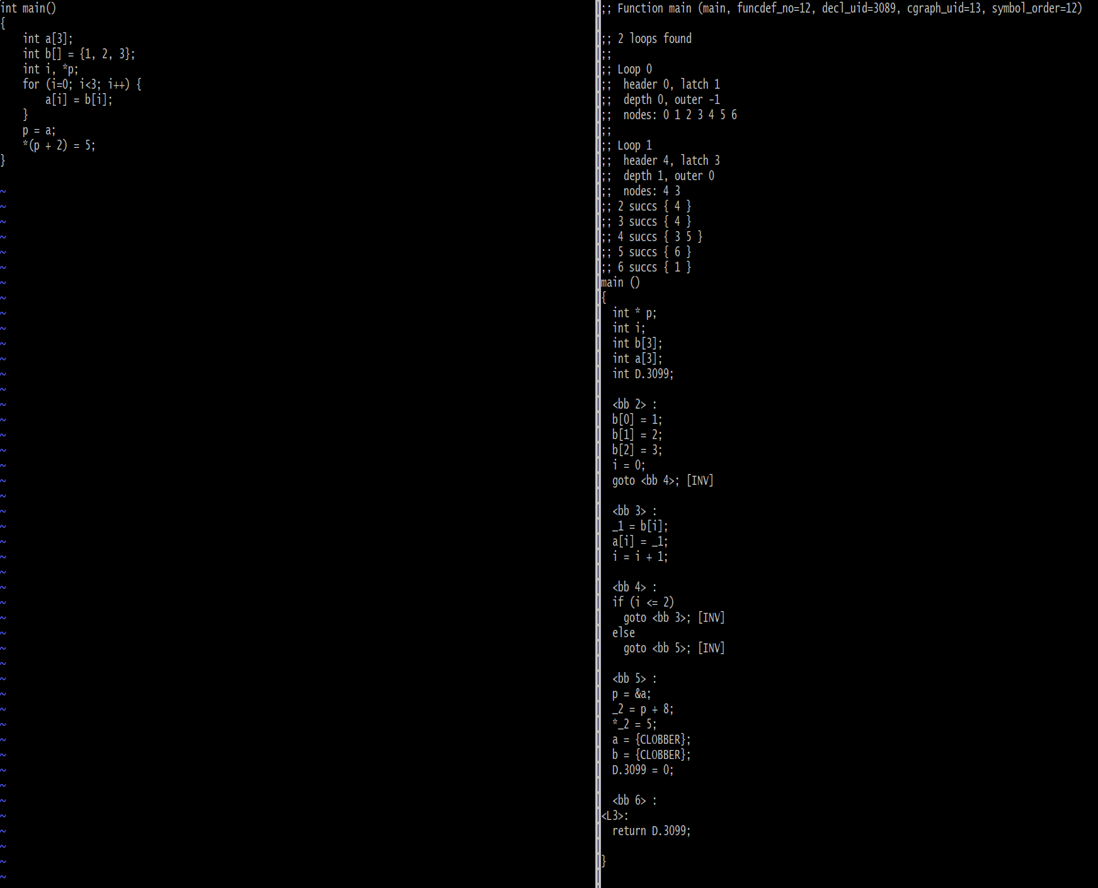
initial basic block, the body of the loop, finally the increment

part at the end of the loop body, and the conditional jump

back to the start of the loop.

A-4 SOLUTIONS:

(ARRAYS AND POINTER)



1. Array Initializations is same as scalar initialization .It is split into

Multiple statements that assign values to individual array elements.

2 As GIMPLE can perform only one operation per statement it can’t

read and write to array locations at the same time (as array

accesses involve offset calculation). Thus, these are split into

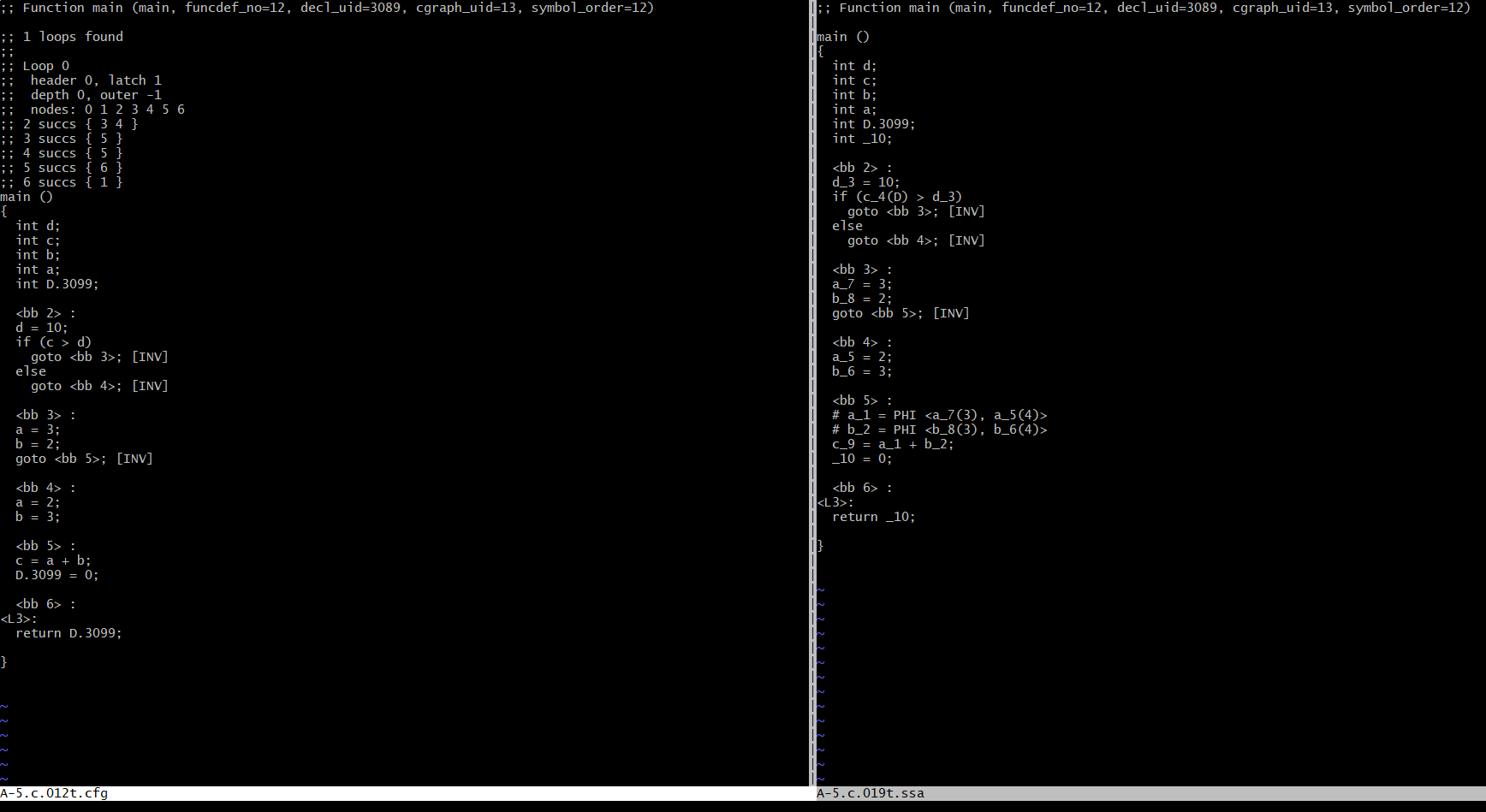
separate read and write statements by introducing a temporary.

3 Clobber statements indicates that the scope of that particular address

escaped variable has ended.

A-5 SOLUTIONS:

(STATIC SINGLE STATEMENTS)



1 Local variables are suffixed with a unique number which represents

the assignment. `d\_3' refers to the usage of variable ‘d’

Variables that have to be used before any coding or assignment

are suffixed with ‘D’ to indicate that assignment was implicit in

This happens to both uninitialized local variables and for

formal parameters. For example, variable `c' is

used as `c\_4(D)' in the `if' condition.

2 A PHI node merges assignments from different paths and creates a new

assignment. For example:

# a\_1 = PHI <a\_5(3), a\_7(4)>

indicates that if the control reaches here from basic block 3 then

use the value of a\_5; otherwise if the control reaches here from

basic block 4 then use the value of a\_7.

3 A switch statement can result in a merge of more than two paths (and

hence possibly more than two assignments). Consider:

int main()

{

int a, b;

switch (a)

{

case 1:

b = 2;

break;

case 2:

b = 3;

break;

case 3:

b = 4;

case 4:

b = 5;

break;

case 5:

case 6:

b = 6;

break;

default:

b = 7;

}

return b;

}

In this case the PHI node at the end of the switch statement will

merge multiple assignments such as:

# b\_1 = PHI <b\_4(3), b\_5(4), b\_7(6), b\_8(7), b\_9(8)>

A-6 SOLUTIONS:

(GIMPLE TO C CONVERSION)

The code which I converted from gimple to C is:

‘ The Program to calculate area and circumference of circle:

#include<stdio.h>

int main() {

int rad;

float pi = 3.14, area, ci;

printf("Enter the radius of circle: ");

scanf("%d", &rad);

ci = 2 \* pi \* rad;

printf("The circumference of circle is: %.2f\n", ci);

area = pi \* rad \* rad;

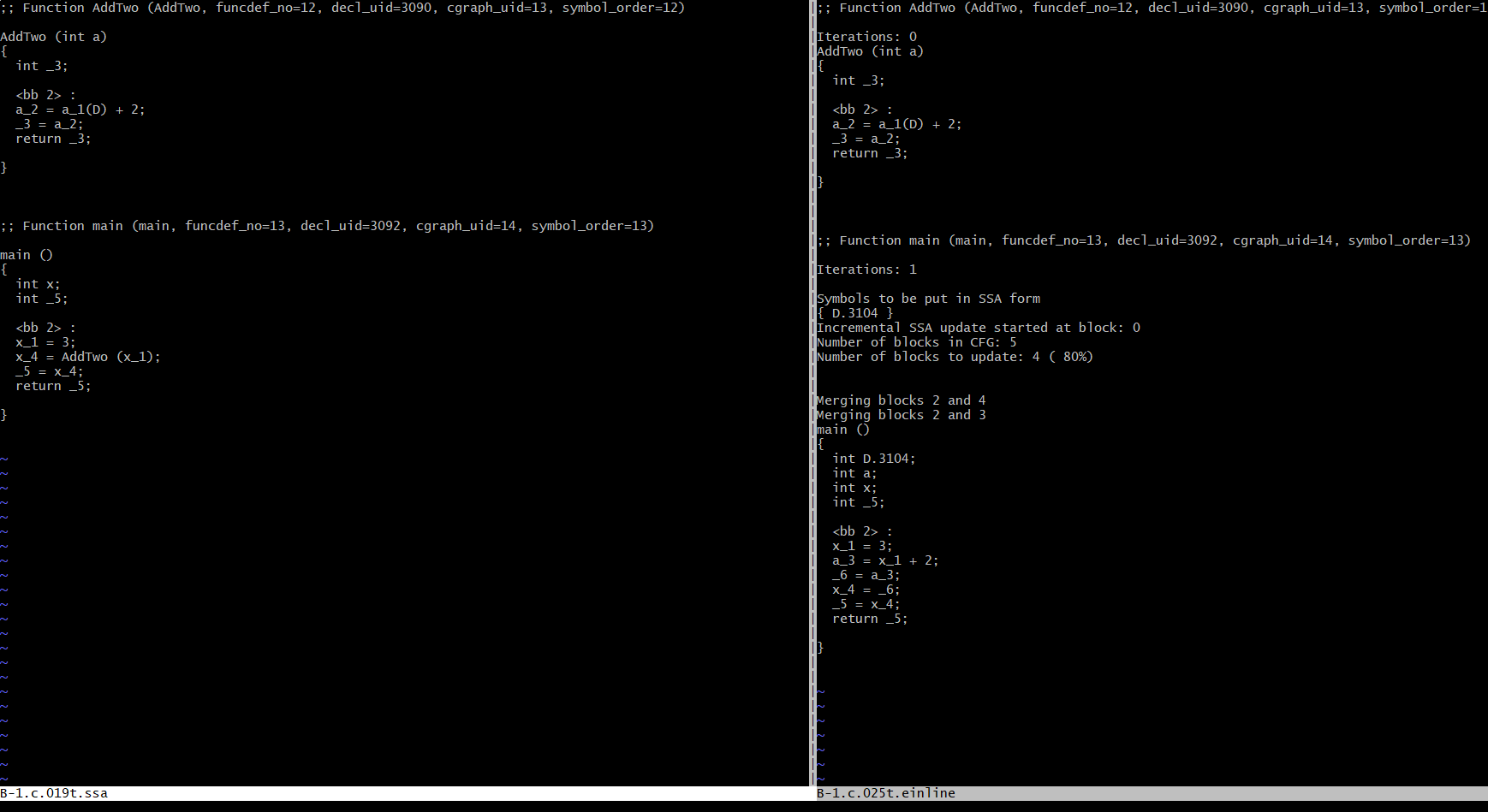
printf("The area of circle is: %.2f\n", area);

return 0;

}

B-1 SOLUTIONS:

(FUNCTION INLINING)



1 There is possibility of that function body to be from another file so it can

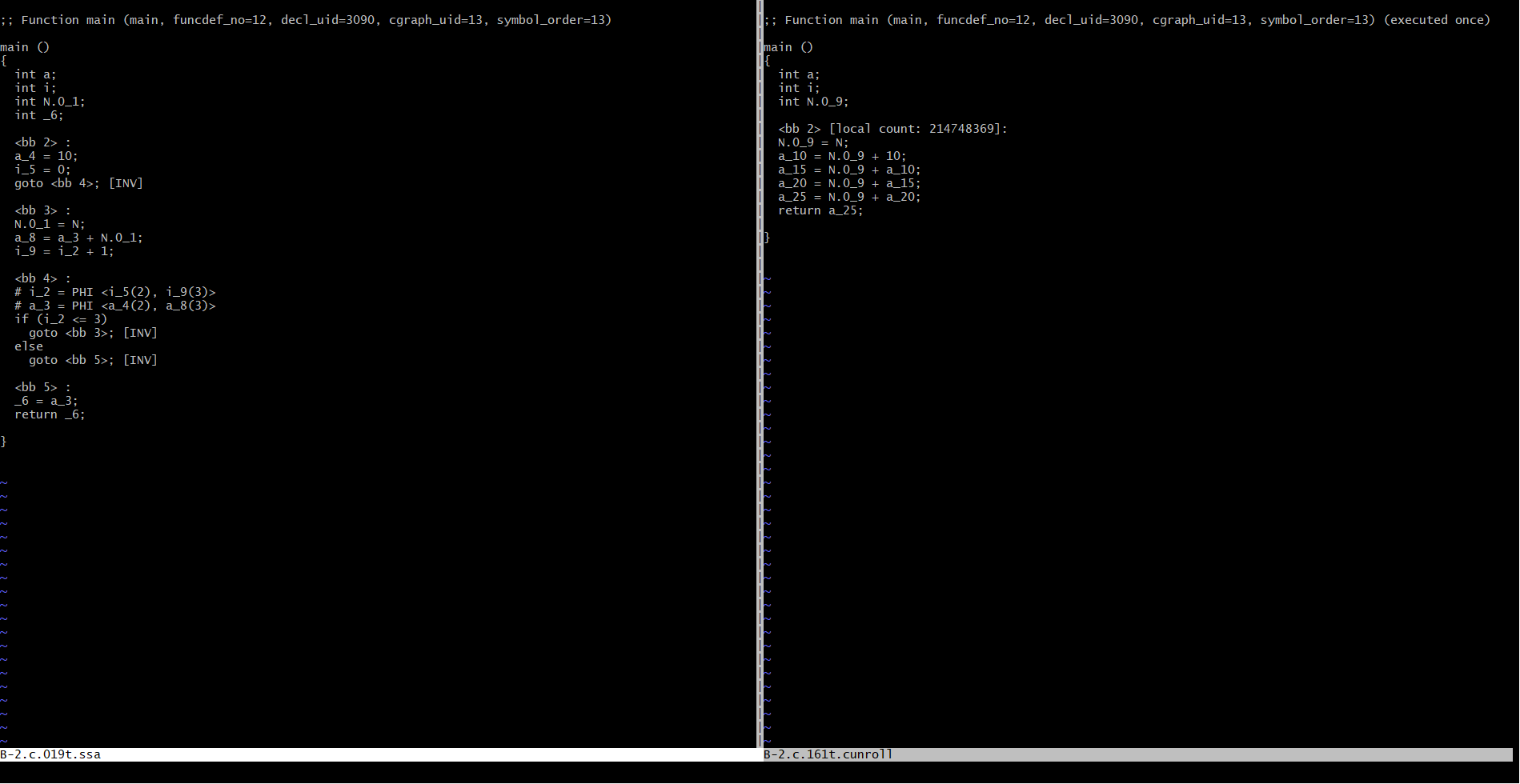
Exist and so compiler may not know about this before link time.

2 If the function was declared with static keyword and if there is no caller

. within the file then the function body or code will disappear.

B-2 SOLUTIONS:

(LOOP UNROLLING)



1 The loop will be unrolled and constant propogation will allow GCC to

compute the final value of `a'. Hence the loop will be removed

entirely.

main () main ()

{ {

int a; <bb 2>:

int i; return 14;

int D.1593;

}

<bb 2>:

a\_3 = 10;

i\_4 = 0;

goto <bb 4>;

<bb 3>:

a\_5 = a\_2 + 1;

i\_6 = i\_1 + 1;

<bb 4>:

# i\_1 = PHI <i\_4(2), i\_6(3)>

# a\_2 = PHI <a\_3(2), a\_5(3)>

if (i\_1 <= 3)

goto <bb 3>;

else

goto <bb 5>;

<bb 5>:

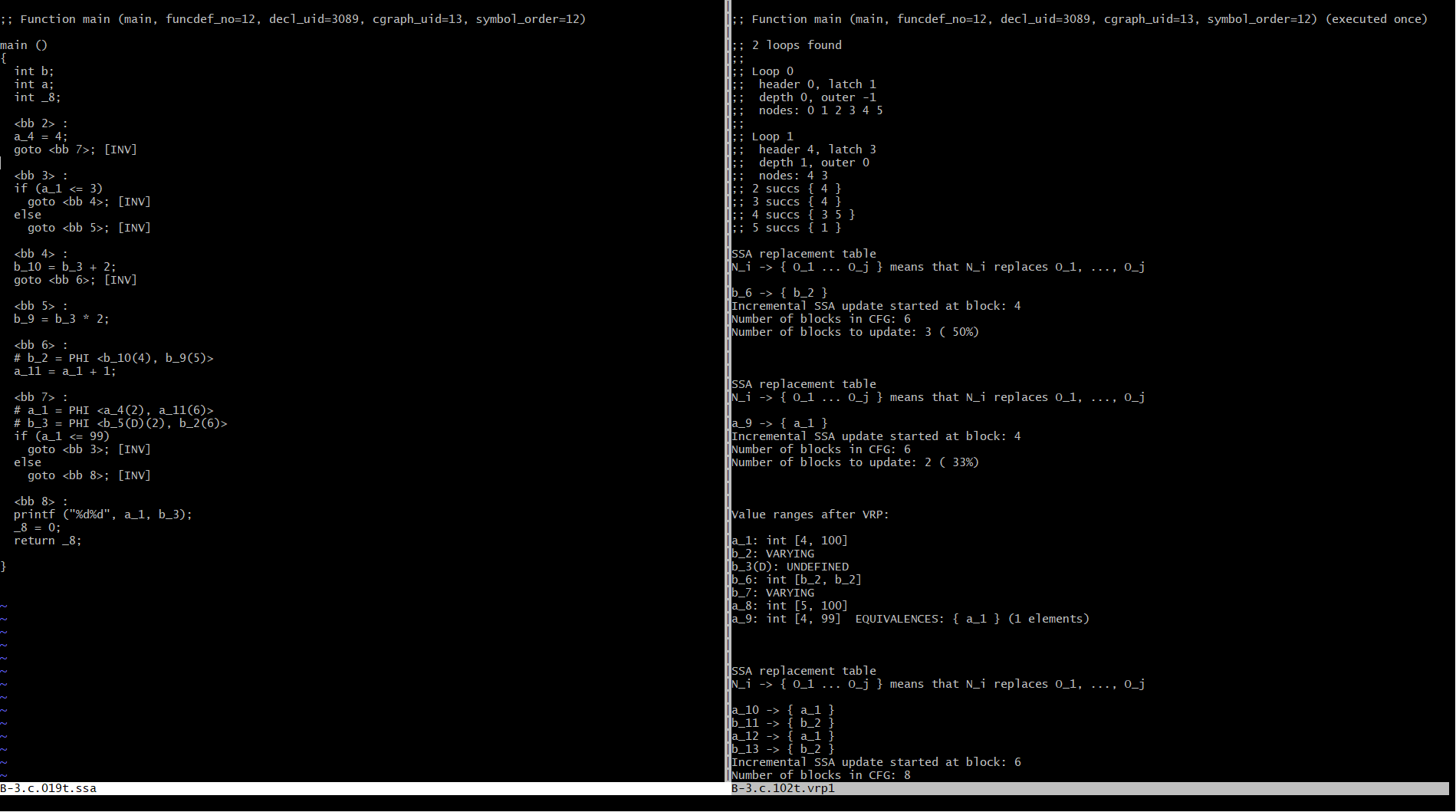
D.1593\_7 = a\_2;

return D.1593\_7;

}

B3 SOLUTIONS:

(VALUE RANGE PROPAGATION)



1 Variable `a' can take the values 4 to 99. Since, the `if' condition

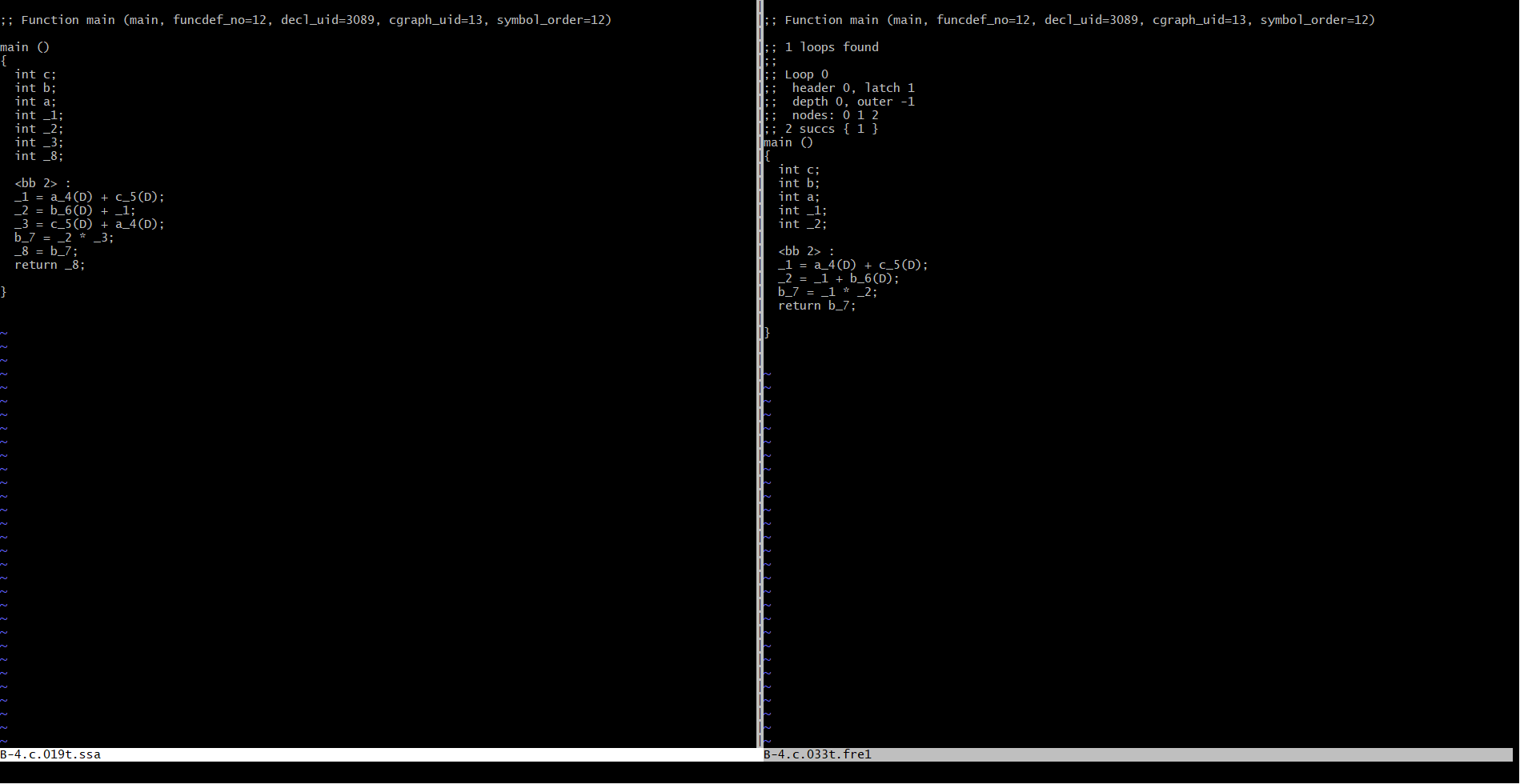
will never be true for the give loop bounds, [b = b + 2] is never

executed. Therefore, the optimized code has eliminated this program

statement.

B-4 SOLUTIONS:

(Common Subexpression Elimination)



1 The original code showed two computations, but the FRE pass was able

to reuse the temporary variable which stored the result of the

intitial computation of (a + c). Thus, one line of execution was

saved, and so was the use of an extra variable.

2 Check the file B-4.c.\*.optimized. This is the last GIMPLE

intra-procedural pass. If it is same as the output of FRE, then no

more optimizations have occured.

3 "return b" was required to make the computation of variable `b'

meaningful. If function `main' is going to "return 0" and there is

no other output, GCC will realise that the entire computation of

variable `b' was dead code and will eliminate its dependencies one

by one until the resulting program reduces to:

int main()

{

return 0;

}