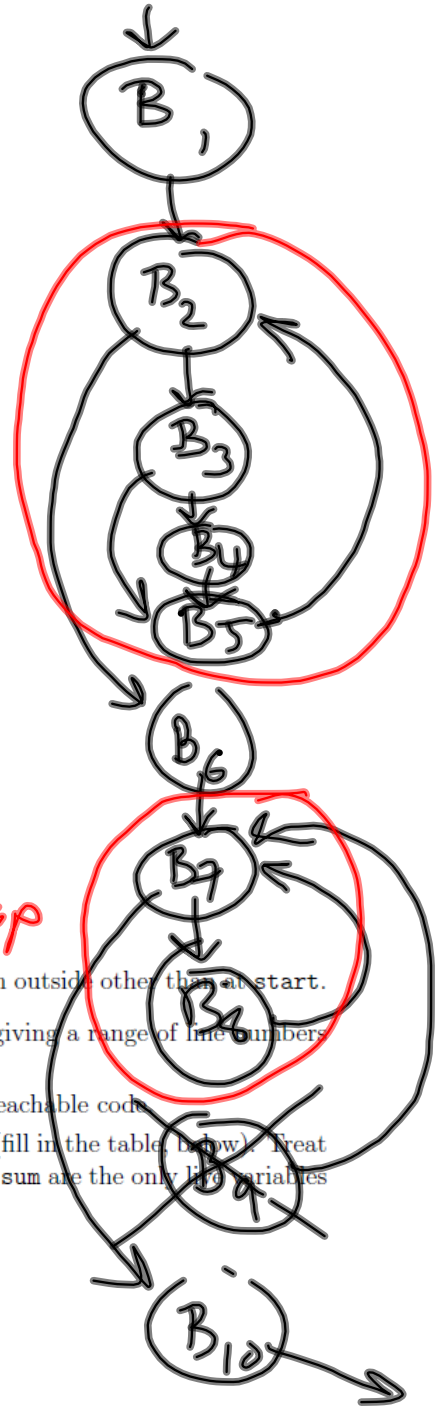
3. The following fragment of 3-address code was produced by a non-optimizing compiler:

```
 1  start:  x := 1
 2          y := 1
 3          sum := x
 4          sum := sum + y
 5  loop:   if x = n then goto out2
 6          t1 := y
 7          t2 := t1 * t1
 8          t3 := t1 + x
 9          if t3 < n then goto skip
10          t3 := t3 - n
11  skip:   sum := sum + t3
12          y := y + 1
13          goto loop
14  out2:   x := sum
15          sum := x + 1
16          x := x + 1
17  out1:   x := x - 1
18          t1 := x
19          print t1
20          if x = 0 then goto out
```




```
21          goto out1
22          goto out1
23  out:    no_op
```

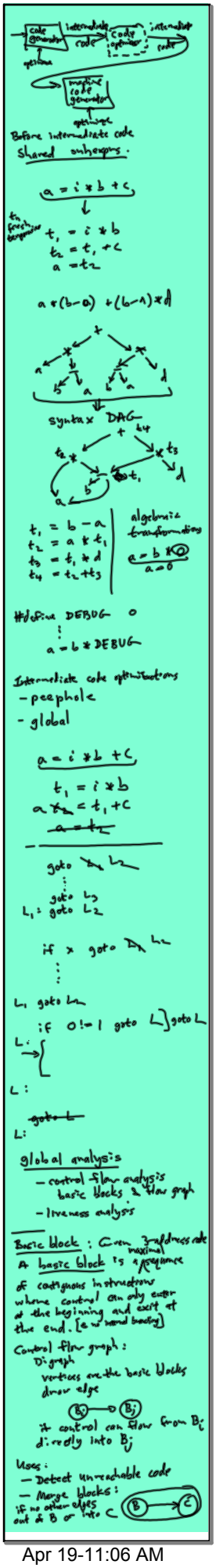
Assume that there are no entry points into the code from outside other than at `start`.

(a) Decompose the code into basic blocks $B_1, B_2, \ldots$, giving a range of line numbers for each.

(b) Draw the control flow graph, and describe any unreachable code.

(c) For each control point, list which variables are live (fill in the table below). Treat the array `a` as a single variable. Assume that `n` and `sum` are the only live variables immediately before line 24.

Before line	Live variables
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	


code generator
intermediate code
code optimizer
source program
machine code generator
target program
Before intermediate code
Shared subexpr.
a = i * b + c
t1 = i * b
t2 = t1 + c
a = t2
a * (b - c) + (b - c) * d
syntax DAG
algebraic transformations
a = b * 0
a = 0
#define DEBUG 0
a = b * DEBUG
Intermediate code optimizations
- peephole
- global
goto L2
goto L3
L1: goto L2
if x goto L1
goto L2
if 0 != 1 goto L1 goto L2
L1:
goto L1
L1:
global analysis
- control flow analysis
basic blocks & flow graph
- liveness analysis
Basic block : Given 3-address code
A basic block is a maximal sequence
of contiguous instructions
where control can only enter
at the beginning and exit at
the end. [& w/ no halting]
Control flow graph :
Digraph
vertices are the basic blocks
draw edge
if control can flow from Bi
directly into Bj
Uses :
- Detect unreachable code
- Merge blocks :
if no other edges
out of B or into C

Detect loops

A loop is a set L of vertices in a control flow digraph satisfying

1. L is strongly connected
   [can follow a path in L from any node in L to any other node in L]
2. L has a unique entry point

An inner loop is a loop that has no proper subloops.

most execution time is spent in inner loops.
concentrate optimization here.

Liveness analysis

$$x = y * z$$

x is set; y & z are used

at any given point in a program: a variable is live (or alive) if there is some control flow path to a spot where that variable is used with no intervening set of the variable.
Otherwise, var is dead.

$$x = x * 2$$

after; before

~~x = y * z~~ → x = 6

can remove if x is dead