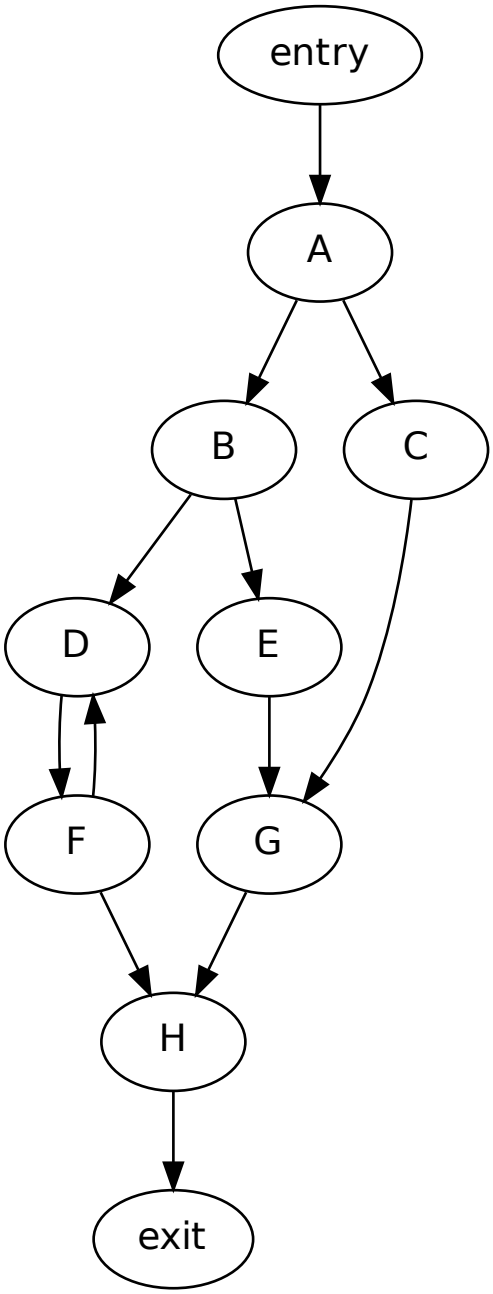
# Homework 8 Solutions

# 1 Control Flow

## 1.1 Graph



### 1.2 Dominators Starting With A

| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| | A | N | N | N | N | N | N | N |
| 1 | A | A,B | A,C | A,B,D | A,B,E | A,B,D,F | A,G | A,H |
| 2 | A | A,B | A,C | A,B,D | A,B,E | A,B,D,F | A,G | A,H |

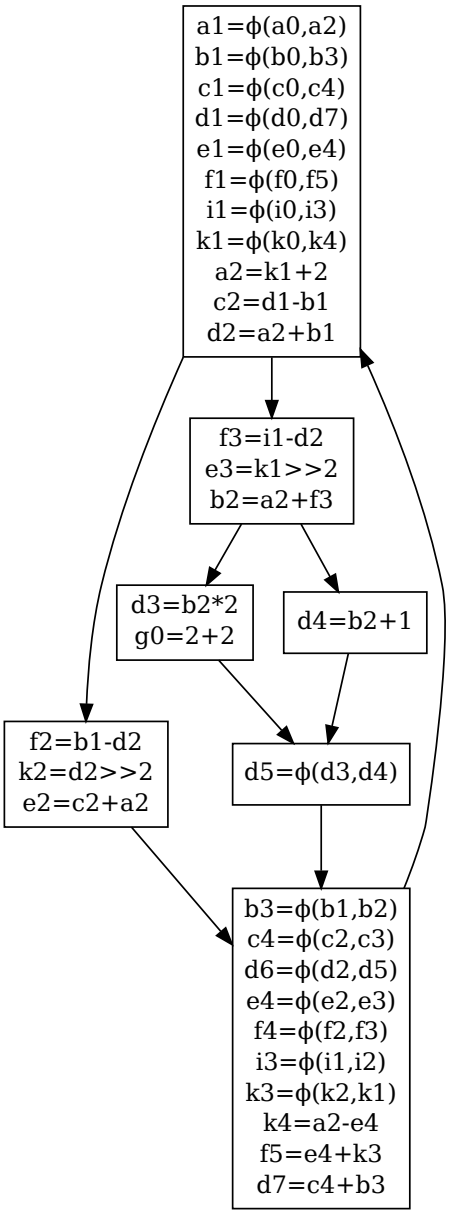
### 1.3 Dominators Starting With H

| | H | G | F | E | D | C | B | A |
|---|---|---|---|---|---|---|---|---|
| | H | N | N | N | N | N | N | N |
| 1 | H | G | F | E | D | C | B | A |
| 2 | H | G | F,D | E,B | D,B | C,A | B,A | A |
| 3 | H | G | F,D,B | E,B,A | D,B,A | C,A | B,A | A |
| 4 | H | G,A | F,D,B,A | E,B,A | D,B,A | C,A | B,A | A |
| 5 | H,A | G,A | F,D,B,A | E,B,A | D,B,A | C,A | B,A | A |
| 6 | H,A | G,A | F,D,B,A | E,B,A | D,B,A | C,A | B,A | A |

### 1.4 Dominance Frontier

- DF(A) = $\emptyset$
- DF(B) = G,H
- DF(C) = G
- DF(D) = D,H
- DF(E) = G
- DF(F) = D,H
- DF(G) = H
- DF(H) = $\emptyset$

## 2 SSA


a1=ϕ(a0,a2)
b1=ϕ(b0,b3)
c1=ϕ(c0,c4)
d1=ϕ(d0,d7)
e1=ϕ(e0,e4)
f1=ϕ(f0,f5)
i1=ϕ(i0,i3)
k1=ϕ(k0,k4)
a2=k1+2
c2=d1-b1
d2=a2+b1
f3=i1-d2
e3=k1>>2
b2=a2+f3
d3=b2*2
g0=2+2
d4=b2+1
f2=b1-d2
k2=d2>>2
e2=c2+a2
d5=ϕ(d3,d4)
b3=ϕ(b1,b2)
c4=ϕ(c2,c3)
d6=ϕ(d2,d5)
e4=ϕ(e2,e3)
f4=ϕ(f2,f3)
i3=ϕ(i1,i2)
k3=ϕ(k2,k1)
k4=a2-e4
f5=e4+k3
d7=c4+b3

# 3 SSA II

## 3.1 Critical Instructions

writeInt, cbr, jump, exit

## 3.2 Mark

```
writeInt x1
writeInt y0
jumpI -> B2
loadI 0          => t0
cmp_GTE x0, t0 => t1
cbr t1 -> B3, B4
phi(x0, x2)   => x1
loadI 1          => t2
add x1, t2      => x2
loadI    10   => x0
loadI    2    => y0
```

## 3.3 Rewrite

```
B1:
loadI    10   => x0
loadI    2    => y0

B2:
phi(x0, x2)   => x1
loadI 0          => t0
cmp_GTE x0, t0 => t1
cbr t1 -> B3, B4

B3:
loadI 1          => t2
add x1, t2      => x2

B4:
writeInt x1
writeInt y0
exit
```

# 4 Optimization Passes

## 4.1 Consequence of a compilation pass that creates code that sometimes behaves differently.

This is a failure of not honoring correctness. Could cause bugs and security vulnerabilities.

## 4.2 Consequence of pass that creates code that's slower than the original.

Will create a slower program. Could have very undesirable effects on real-time and high-performance systems.

## 4.3 Acceptable to intentionally have pass that changes the validity of some programs if it preserves correctness of most programs?

No.

## 4.4 Acceptable to intentionally have pass that produces worse performing code on some programs, but speeds up most programs?

Yes.

# 5 Machine-Dependant Optimization

Machine-independant optimizations that can be applied to the internal representations of compilers, and should speed up the majority of programs. Machine-dependant optimizations can be applied to machine-specific representations, and the resulting code will only work on machines that share the target's architecture.

# 6 Analysis/Optimization Scope

## 6.1 Local Scope

A local analysis scope means the analysis/optimization only takes into account instructions inside a basic block.

## 6.2 Global Scope

A global scope means the optimizer takes into account the interactions of all instructions.

### 6.3 Intraprocedure Scope

Intraprocedure scope means the analysis takes into account the interactions of instructions in the same procedure.