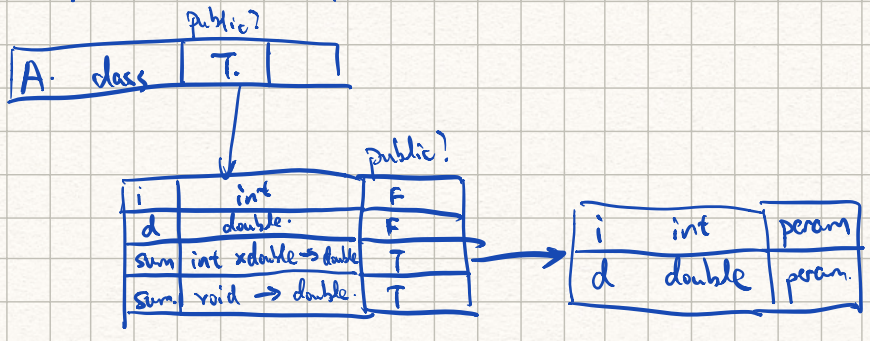
Compiler back end Exercise 5

1. (a)

i. A. {i: int, d: double}. sum1 : int × double → double, sum: void → double.

ii.



(b) (1) type of (10) = int.

typeOf (20) = int.

typeOf (m.sum) = int × int → int.

typeof (d1) = double.

int → double conversion performed automatically / no error.

(2) Similarly type error double → int is not done automatically

(3). typeof (5, 6, +, 7, m.sum) ⟹ typeof (1)

no type error.

rules: 1. identifier declared with data types

2. operands / operations / functions / params shall have correct DT.

3. Assignment have covert w.dt (subtype / matching type)

4. Method
   * Arity
   * parameter type.
   * return type.

2. (a) int tmp declared in a compound statement (inner scope) / not. effect tmp * 2.

(b) no. Context-fuel. / CFG is not context.

(c) hash function (may cause collision).

Scoping implementation is harder with hash tables

↑ multiple table && connection

3. while ((a*b < 100 && a/b >= c) || d.) {
   a = a-1/2*3
}

LOOP
t = T((a*b---||d)
fjump t END
T(a=---)
jump LOOP
END

T((a*b)---c) || d)

⇒ t = T((a*b)--c)
fjump t END. whilecon
t = T(d)
END. whilecon.

⇒ T(a*b)<100 && a/b>=c)
t = T(a*b<100) → t1 = T(a)
fjump t END and. t2 = T(b).
t = T(a/b>=c) t3 = t1 * t2.
END and. t = t3 < 100

t4 = T(c)
t3 = t1 / t2
t = t3 >= t4

T(a = a-1/2*3).
t5 = T(a).
t6 = 1 t6 = t6/t7
t7 = 2.
t8 = 3 t6 = t6*t8
t5 = t5 - t6.

LOOP
t3 = a MUL b.
t = t3 LE 100.
fjump t END and.
t3 = a DIV b.
t = t3 GEQ c.
END and.
fjump t END whilecon.
t = d.
END whilecon.
fjump t END
t6 = 1 DIV 2.
t6 = t6 MUL 3.

t5 = a SUB t6
a = t5
jump LOOP
END

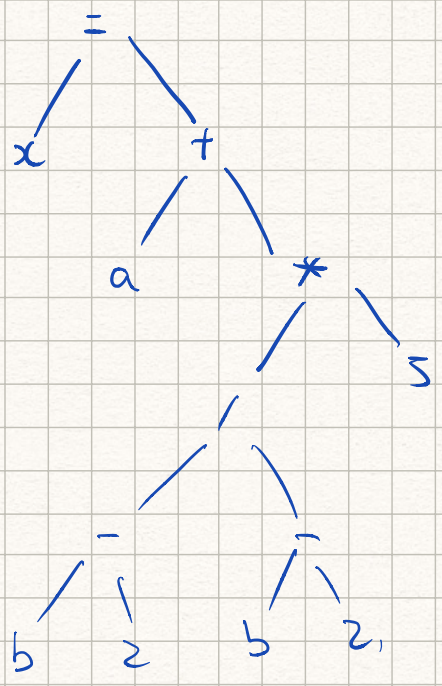
4. repeat {
   <body>
} until (<expr>)

LOOP:
  T(<body>)
  t = T(<expr>)
  tjump t END
  jump LOOP
END:

/ fjump LOOP

Compiler Exercise 6.

1.



t1 = b SUB 2.
t2 = t1 DIV t1.
t3 = t1 MUL 5.
t4 = a ADD t3.
x = t4.

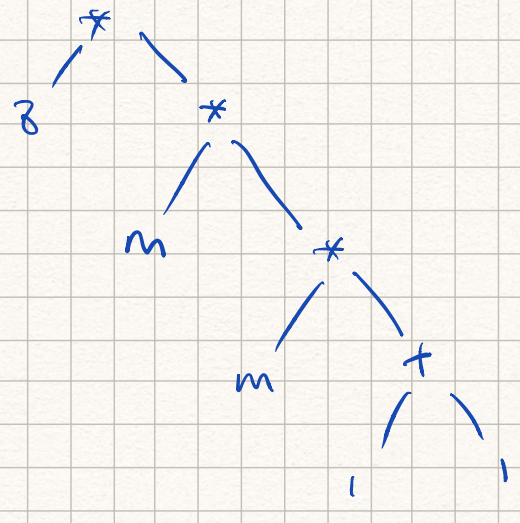
common subexpression

arithemetic identities (counts of bits type)

strength reduction

m*8	m<<3
m*2	m<<1.

Constant folding



t1 = 1 ADD 1
t2 = m MUL t1
t3 = m MUL t2.
t4 = 8 MUL t3
g = t4

3. while ((a*b < 100 && a/b >= c) || d) {

a = a - 1/2 + 3;

}.

LOOP
t = T(a --- d)
fjump t END
T(a --- 3)
jump LOOP
END

→ t = T(a --- c)
tjump T ENDOR
t = T(d)
ENDOR

→ t1 = a*b
t = t1 LE 100
fjump t ENDAND
t2 = a/b
t = t2 GEQ c
ENDADD

t3 = 1 / 2
t4 = t3 * 3
t5 = a - t4
a = t5

↓

LOOP
t1 = a*b
t = t1 LE 100
fjump t ENDAND
t2 = a/b
t = t2 GEQ c
ENDADD
tjump t ENDOR
t = T(d)
ENDOR
fjump t END
t3 = 1/2
t4 = t3 * 3
t5 = a - t4
a = t5
jump LOOP
END

4. Loop unfolding

foo() foo() foo()

adv. less jumps / less predicate check

dis. code size ↑ instruction grow out of register size

Blocking,

```
for (int j=0; j<x-8; j+=8)
    for (int i=j; i<j+8; i++)
        foo()
```

adv. locality. loaded to register.

dis adv. more predicates / jumps

Blocking ??

foo().

or. if it's completely same after change

f. - - - - i 0~x-1.

Code Motion

- Inlining.

adv. reduce costly function calls (stack free ?

dis adv.

Cause code size to grow

```
int main {
    x=7
    y=9
    return 7>9 ? 7:9
```

Constant folding / arithmetic operation.

known result.

```
int main {
    return 9
}.
```

- Inlining (max)

- Unrolling and folding if possible.

param reduction. long func (i)

Inlining. in the loop / Blocking