Unary Operators

OP (single operand)

(−) a   −743   −123   −0.567

Increment operator

++

i = i+1

int i = 3;

++i;

→ i = i+1

i = 4

i++;

└ i = i+1

= 5

OP (single operand)

or

(single operand) OP

Decrement operator

− −

i = i − 1

int i = 3;

−−i;

i = i−1

= 3−1 = 2

i−−; → 2

i = i−1 └→ ①

++a → a=a+1 → pre-increment → increase and use

a++ → a=a+1 → post-increment → use and increase

```
int a=5;
printf("%d", ++a);
```

6

↑ pre

```
int a=9
printf("%d", a++);
```

9

↑ post

```
printf("%d", a);
```

10

a+b

++ +a; → error

```
int a=3;
b=++a++;
```

Error

4++

Lvalue required

b=b++ +a;

b++ +a;

b+ ++a;

int i=3;

→ a=++i; a=i++; ←

a=4 a=3

i=4 i=4

i=3

a= ++i * ++i * ++i;

6 5 4

= i * i * i

= 6 * 6 * 6

= 216

a= ++i * ++i * ++i;

25

6 5 4

64

120

150

216

Undefined

Associativity	Precedence
R→L	Unary operators have higher precedence over the binary Arithmetic operator

→ Compiler dependent

→ Undefined.

① a = i++ + i; ←

② a = i++ + i++ + i; ←

sequence point

Increment / Decrement operator

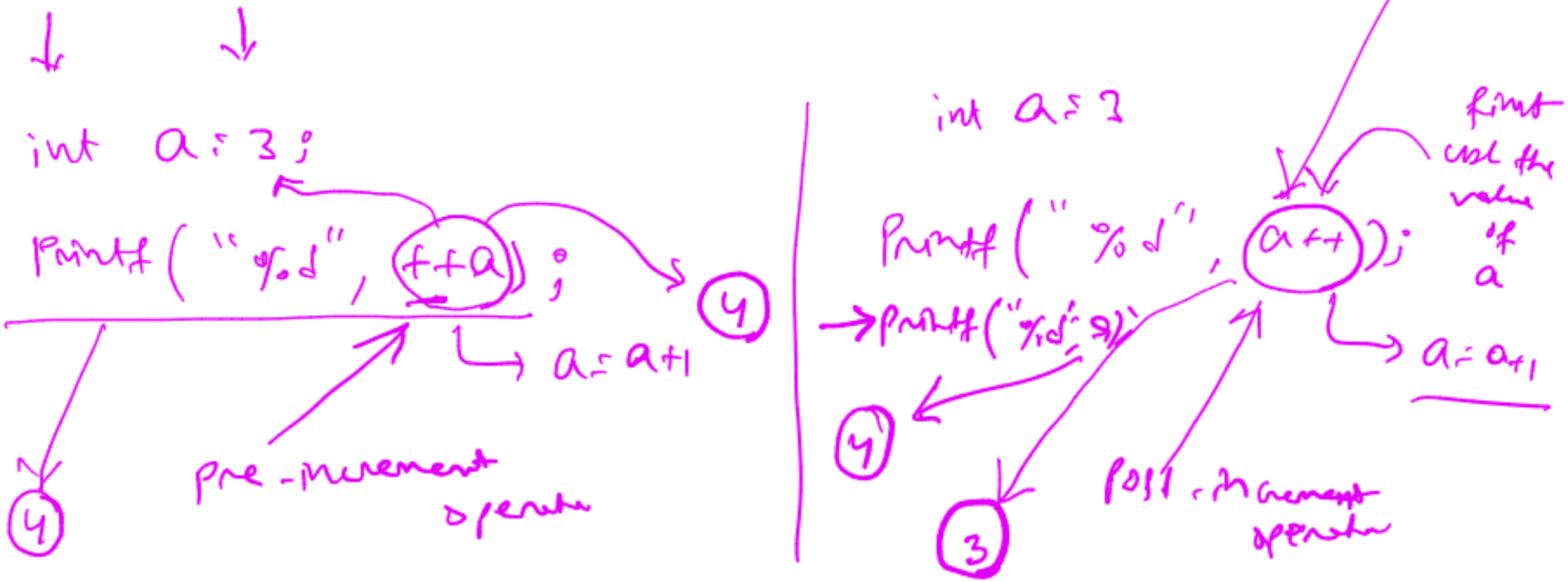
++ → increment operator → $a = a+1$

-- → Decrement operato → $a = a-1$

$a=4$ ++a → $a=a+1$ ⇒ 5

--a → $a=a-1$ ⇒ 3

Pre / Post



```
int a=3;
printf("%d", ++a);
```

```
int a=3
printf("%d", a++);
printf("%d", a);
```

Pre / Post decrement

a = 4

printf("%d", --a);

→ pre-decrement

→ a = a-1

③

a = 4

printf("%d", a--);

④

→ post-decrement

→ a = a-1

④

→ 3

variable

Associativity

R → L

Precedence

Unary operator has higher precedence than the binary arithmetic operator.

int i=3

a=++i;   a=i++;

→ a=?4   a=?3 ←

→ i=?4   i=?4 ←

int i=6

a=--i;   a=i--;

a=5   a=6 ←

i=5   i=5 ←

i=3;

a= ++i * ++i * ++i ;

5   4

= 6 * 6 * 6;

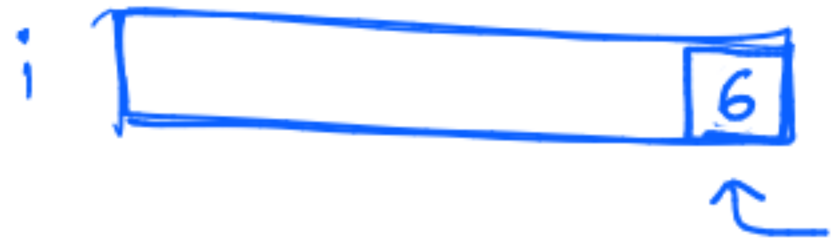
= 216

undefined ←

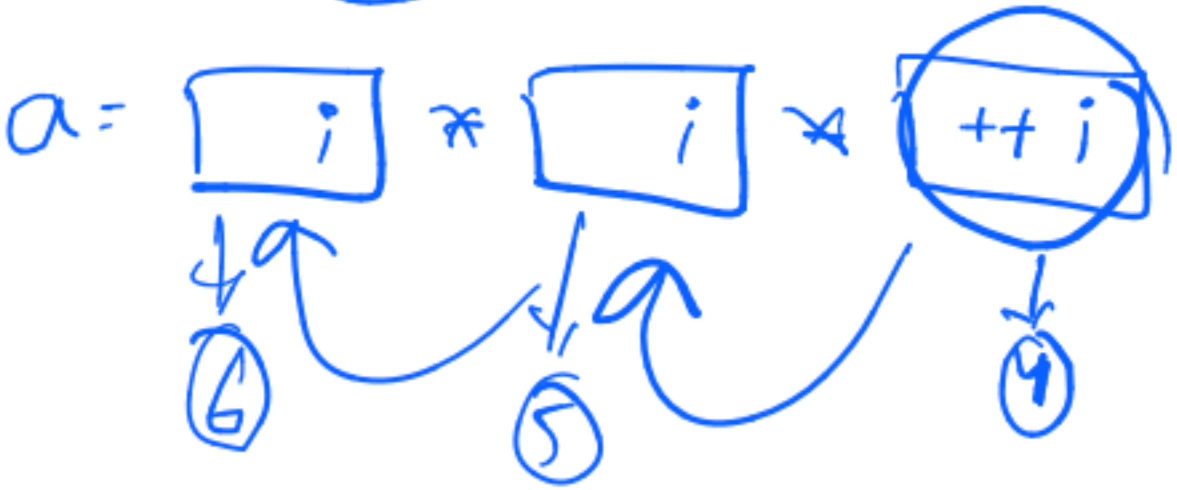
a=64

a=120

R→L



i=3



i * i * i

6 * 6 * 6 = 216

Compiler dependent

① a= i++ + i ; ←

② a= i++ + i++ + i ; ←

sequence points

a = b + c ;

sequence point

compiler guarantees finish all the evaluation before the sequence point.

i=3

a= ++i * $\boxed{\text{++i * ++i}}$;

⑤ ④

216

150

undefined

a= b+c ;

= ++i + ++i ;

i don't know

→ undefine

If you are applying more than one operation on a single variable in the same expression, then the result is undefined.

$a = ++i * ++i + --i;$ ⟶ undefined.

$a=3$

$a = \underbrace{a++}_{3} + \underbrace{a}_{4};$ ✓

$a = \underbrace{++a}_{4} + \underbrace{a}_{4};$

---

int a=5

⟹ $b = \underline{--(--a)};$

$--4$

$a = ++5 + ++5$

→ **Error** Lvalue required

increment / decrement operator cannot be applied on constants.

$--5$

$++a++$ X $--10$ X $--a$

sizeof operator

This operator will give you the size of a variable or data type.

K = sizeof(int); → 2 bytes

sizeof( );

int a;

sizeof(a); → 2 bytes,

Cast is also a unary operator

$\underbrace{(\text{int})}_{\text{casting operator}}\underbrace{(a+b)}_{\text{float}}\ \%\ 5$

a is int

b is float

a % b

a % (int) b

(int)(b % a)

R→L

int a = 4

int b = ++a;

b = 5

a = 5

int b = a++;

b = 4

a = 5

int a = 3

int b = ++a * ++a * ++a;

120

64

150

216

→ Undefined

++a + ++a + a;

→ Undefined

Sequence points

Compiler guarantees to finish all the evaluation before the sequence point.

a = b + c ;

int b = ++a + ++a + a ;

If you are applying more than one operation on a single variable in the same expression in between sequence points, then the result is undefined.

$\underline{a = 1}$

Unsequenced

$b = a + \boxed{a++} + a + a;$ → ⑥ ⑤ $\boxed{7}$ ✓

2 1 2 2 → ⑦

$\boxed{b = a + a + a + (a++) + a;}$

2 2 2 1 2 → ⑨ ⑥

9

6

5

4

undefined ✓

Compiler dependent

$a = 1,\ b = 2$

$K = a + ++b;$

$K = \boxed{4}$ ✓

$K = a + ++a + a++;$

Unsequenced

Single compiler

↓

same logic

↓

you will get different result

int k, a=4;

$k = ++\boxed{3};$ ⟶ Error

↳ constants

$k = \boxed{++a}++;$ ⟶ Error

sizeof operator

This operator will give you the size of a variable or data type.

k = sizeof(int);

2

sizeof (a) ;

int a ;

float a ;

2

4

int a = 3 ;

k = sizeof(++a) ;

k = ? 2 ✓

a = ? 4 ✗

3

NO

int 2

cast → Unary operator

Relational and logical operators

<	>	<=	>=
↓	↓	↓	↓
lt	gt	lte	gte

Left

>

==	!=
↓	↓
equal to	not equal to.

Right

L → R

a < b

└→ True → 1

└→ false → 0

a=10 , b=15

a >= b → False → 0

b <= a → false → 0

a <= b → True → 1

a == b

└→ True → 1

└→ false → 0

a != b

└→ True → 1

└→ false → 0

a=9, b=10

a == b → 0

a != b → 1

char a = 'w';

$a == 119 \longrightarrow \underline{1}$

char $\underline{a}$ = 'A', $\underline{b}$ = 'B';

$\underline{a >= b} \hookrightarrow 0$

$a <= b \hookrightarrow \underline{1}$

$\underline{'A'} >= \underline{'B'}$

$\downarrow \qquad \downarrow$

$\underline{65 >= 66}$

$\hookrightarrow \boxed{0}$

Logical operators

&& And ← Higher

|| OR ← Lower

L → R

expr 1 | logical operator | expr 2

0 or 1 0 or 1

a = 10, b = 15

$\underset{1}{a}\ \&\&\ \underset{1}{b} \rightarrow \boxed{1}$

If you have a non-zero value as the result of any expression, then consider it as 1.

True → 1

False → 0.

And

Expr 1	Expr 2	Result
True	True	True
True	False	False
False	True	False
False	False	False

OR

Expr 1	Expr 2	Result
True	True	True
True	False	True
False	True	true
False	false	False

a=100 , b=200 , c

c = (a==100) || (b>200)

c = ? → 1

expr1 && expr2

false

True

a = 3, b=0;

L → R

c = ++b || ++a;

c = ? → 1 ✓

b = ? → 1 ✓

a = ? → 4 ✗

3

short circuit

a = -3

++a

a = a+1

= -3+1

= -2