Header files

- iostream
- iomanip → setw()
- math
- algorithm
- vector
- queue

.
.
.

cp

bits/stdc++.h

Namespaces

Java

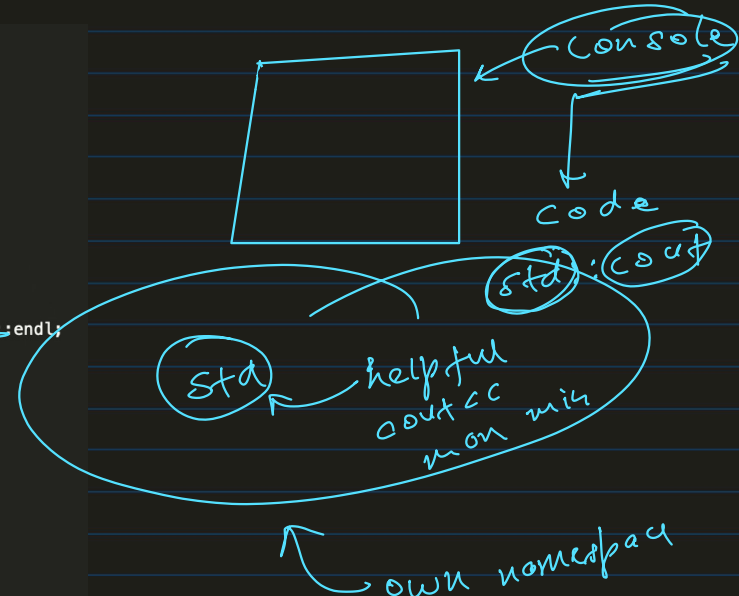
↓

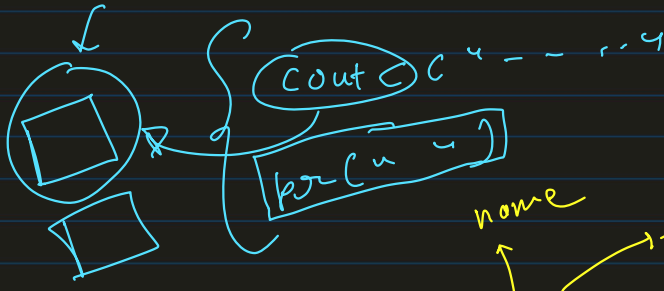
Internal
Personal
Java

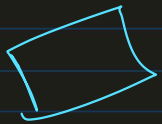
```
namespace std

namespace A {

}
```

```
namespace std;
cout << " Y";
```

```cpp
#include<bits/stdc++.h>

void FIO() {
    std::ios_base::sync_with_stdio(0);
    std::cin.tie(0); std::cout.tie(0);
#ifndef ONLINE_JUDGE
    freopen("input.txt", "r", stdin);
    freopen("output.txt", "w", stdout);
#endif
}


namespace alka {

void cout(int a) {
    std::cout << a << " is present" << std::endl;
}

std::string name = "Alka Gupta";
}

int main() {
    FIO();
    using namespace alka;
    cout(1);
    std::cout << "Hi" << std::endl;
    std::cout << name << std::endl;
    return 0;
}
```







```
void print(int a)
```

name

function sign

return data type

parameter

Main defination

→ compiler's girlfreind

```
int main ( ) {
→
→
→
→
    return 0;
}
```

[f1

[f2

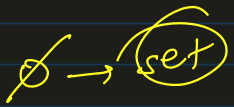
[f3 → segmentation fault

```
int main(){
    f2();
    f1();
    return 0;
}
```

Data Type

→ primary / pre-defined / atomic

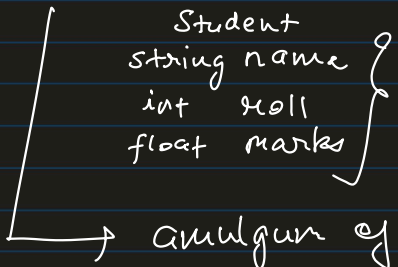
- → int
- → char
- → ~~String~~
- → Boolean
- → float
- → double
- → void



→ User defined

- → class
- → structure
- → Enum
- → Typedef

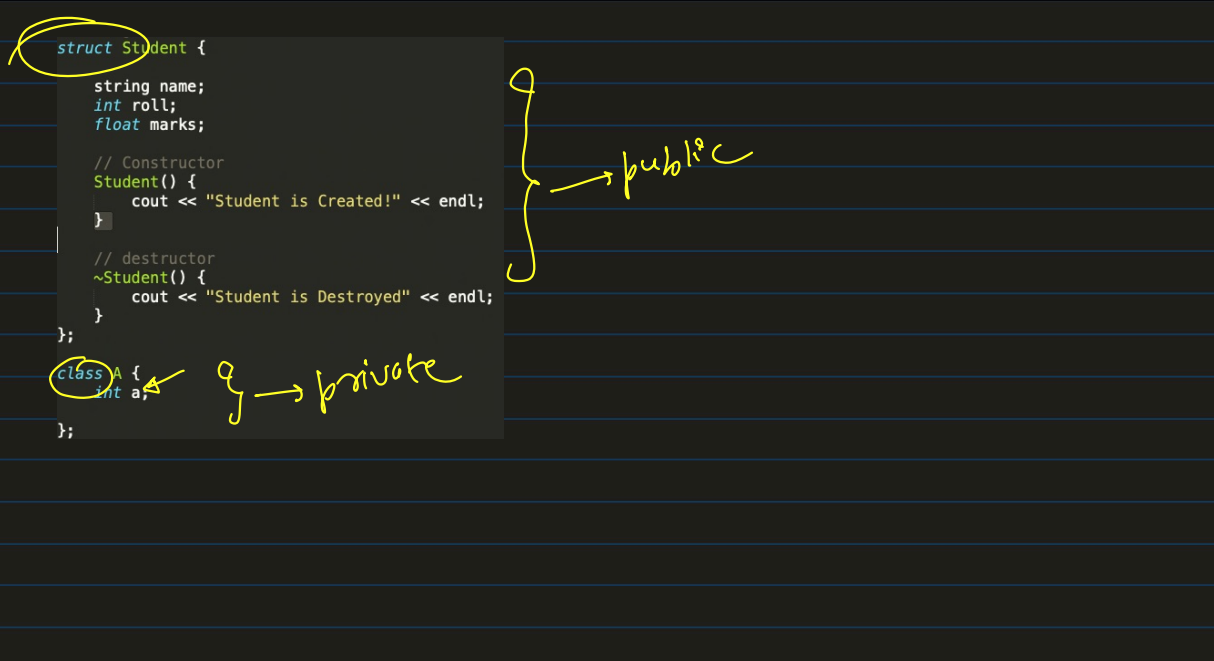
structure



Class

public private protected

C++20


struct Student {
string name;
int roll;
float marks;
// Constructor
Student() {
cout << "Student is Created!" << endl;
}
// destructor
~Student() {
cout << "Student is Destroyed" << endl;
}
};
class A {
int a;
};
public
private

Operator

operands

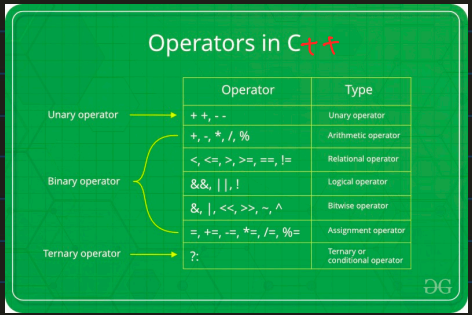
a + b

operator

Expression

Collection of operators & operands

eg :- 4 x 16 / 8



unary → one

→ One operand

a++ , a--, ++a, --a

!a

Binary

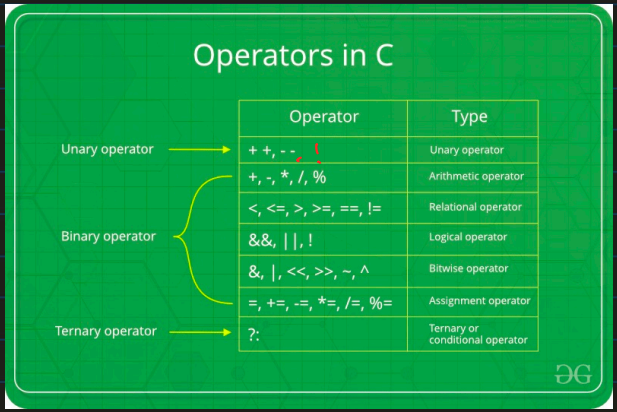
a+b , a+7

a-b

a && b

Ternary

(a>b)? a : b)



Arithematic

+ , - , / , * , %

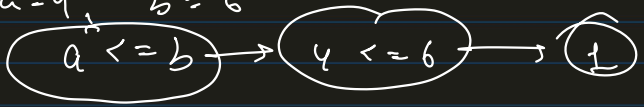
x%on = [0 ..... n-1]

Relational

↳ compare

< , <= , > , >= , == , !=

a=4, b=6



int a = 6 float b = 6.0

a == b

a != b → 0

logical

↳ logic

&& / and

|| / or

! / not

They combine conditions

a = 6 , b = 5, c = 10

a >= b && b <= c

6 >= 5 && 5 <= 10

1 && 1

1

1 || 0 = 0

!1 = 0 !0 = 1

C++ → +ve number, -ve number

true / 1

C++ → 0

↳ false

(4 && 0)

F

(4 || 0)

T

(4 && 6)

T && T

T

(-0 || +0) → -0 == +0

↳ C++

Javascript → -0, +0, 0

while(-1) {

T

Bitwise ★★

&	bitwise	AND
|	"	OR
<<	"	left shift
>>	"	right shift
~	"	Not
^	"	xor

a = 6 b = 4

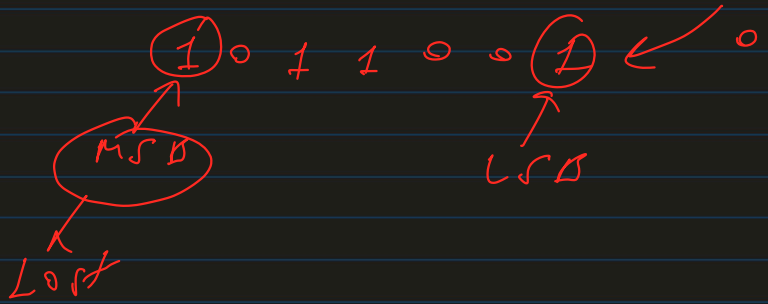
a & b ⟶ a = 6 = 1 1 0
b = 4 = 1 0 0
1 0 0

a & b = 6 & 4 = 110 & 100 = 100 = 4

a | b a = 6 = 1 1 0
b = 4 = 1 0 0
1 1 0 ⟶ 6

<< → left shift

10<<1 → 1010<<1 → 0101



Right shift >>

$9 >> 2 \rightarrow 1001 >> 2$

0100

↳ 0010

Not

$\sim 6 = \sim(110) = (001)$

$\sim 10 = \sim(1010) = (0101) = (101) = 5$

XOR (^)

Truth Table

A	B	A^B
0	0	0
0	1	1
1	0	1
1	1	0

4 ^ 6

```
  1 0 0
^ 1 1 0
-------
  0 1 0
-------
  2
```

Assignment operator (=)

Language Expression ⟶ Right to Left

operator ⟶ Associativity

a = 4

a += 4 ⟶ a = a + 4

4 + 4

a -= 4 = a = a - 4 ....

a %= 4

a = a % 4

a &= 4

↳ a = a & 4

Ternery operator ?:

```
if( x>0){
    val=x
}
else{
    val=-1
}
```

```
val= ( (x>0) ? x : -1)
```

```
( (/* condition */) ? val_true : val_false )
```

type must be same

int num = 56

1 : 0

bool isEven = ((num%2==0) ? true : false)

Precedence & Associativity

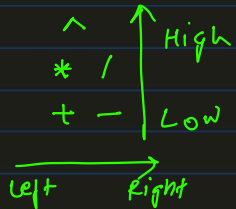
It is used to decide which operation will be performed first (order of operations).

Expression

$10 \times 5 + 6$

$50 + 6$

56



$10 + 5 - 6 + 20$

$15 - 6 + 20$

$9 + 20$

29

funccall → priority high

```
int getnum() {
    return 1;
}
```

Table of precendce & Associativity

Category	Operator	Associativity
Postfix	() [] -> . ++ - -	Left to right
Unary	+ - ! ~ ++ - - (type)* & sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >>	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %=>>= <<= &= ^= |=	Right to left
Comma	,	Left to right

f = a & b * getNum() / d && f

funcn call

f = (a & b) * (d / c)

Typecasting / Type conversion

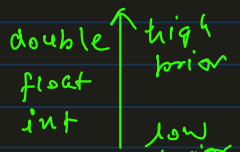
→ Implicit Conversion → अपने आप (compiler)

When there is a mixture of data types in an expression, then all the lower data types are implicit/ automatically converted into higher order data type present in the expression

eg:-

```
float a = 1.5
int   b = 2
float c = a + b
```

3.5 ← 2.0 + 1.5



int m1 = 90, m2 = 90, m3 = 85;

(1) float perc = (m1 + m2 + m3) / 300 * 100

= 0.0

(2)
```
float perc = (m1 + m2 + m3) / 300.0 * 100;
```

= 88.3333

double
float
int

```
char ch = 'A';

cout << 5 + ch << endl;
```

Explicit Conversion

When a data type is molded into another (if possible) forcefully

eg:-

(1) `float here = (float) m1 + m2 + m3) / 300 * 100`

(2)
```
char ch = 'A';
int ord = (int) ch
```

(3)
```
float a = 4.5 , b = 6.5
int ans = (int) a + (int) b
```