__/__/____

CO-4 Operator Overloading.

* Operators which can be OVERLOADED.

BINARY ⇒
1. Arithmetic: +, -, *, /, %, +=, -=, *=, /=
2. Relational: <, <=, >, >=, !=, ==
3. ~~Conditional~~ logical: &&, ||, !
4. Bitwise: &, |, ^, ~, <<, >>

USING MEMBER AND FRIEND FUNCN.

5. Mixed mode arith.: ob1 = ob2 + 5.
6. →
7. ( )
8. [ ]
9. cin>> cout<<
10. conversion op.: BDT & UDT
11. new, delete.

UNARY ⇒
12. pre/post inc.

* Operators which can't be Overloaded.
1. sizeof()
2. ::
3. . (~~dereferencing~~) (dot)
4. * (pointer deref.)
5. ?: [conditional op.]

```
class AB
{ int a;
  public:
    AB (int x = 0)
    { a = x;
    }
    AB operator+ (AB &ob)
    { AB t;
      t.a = a + ob.a;
      return t;
    }
```

```
int main()
{ AB a1(5), a2(6), a3;
  a3 = a1 + a2;
  a3.disp();
```

a3 = a1.operator+(a2);

dummy parameter → POST

__/__/____

```
                    int
AB  operator++ ( )
{   AB t ;
    t.a = ++a;
    return t;
}

AB  operator ++ (int )                    o1 = 5 ,
{   AB t ;
    t = *this ;                  ──→  o2 = (o1++) ;
    (this→a) ++ ;                          ⇓
    return t ;                       o2 = 6 ,  o1 = 5
}
```

```
    If
AB t ;
(this→a) ++ ;
t = *this ;
return t ;
```

```
friend  AB operator++ ( AB &ob , int )
{       AB t ;
        t.a = (ob.a) ++ ;
        return t ;
}
```

POST INCR

~~AC~~
ROBERT
LAH

(11) =3 .!@//01

(2 OR 0)

__/__/____

AB a1, a2;

⇒ if (a1 <= a2)

bool operator <= (AB &ob)
{ if (a <= ob.a) return true;
else return false;
}

⇒ if (a1 && a2)

int operator && (AB &ob)
{ return (a && ob.a);
}

USING FRIEND FUNCTION

friend AB operator& (AB ob1, ~~ob~~ AB ob2)
{ AB t;
t.a = ob1.a & ob2.a;
return t;
}

FOR UNARY OPERATORS

AB operator!()
{ AB t;
t.a = (!a);
return t;
}

o1 → disp()

AB* operator→ ( ) {
    return this;
}

o2 = o1 (5)
⇓
o2 = o1. operator( )(5)

AB operator ( ) (int x) {
    AB t;
    t.a = a+x;
    return t;
}

ob2 = ob1 +5 ⟹ ~~ob1 =~~ ob2 = ob1. operator + (new(5));

AB operator + (AB x)
{ AB t;
    t.a = a + x.a;
    return t;
}

⇒ ob2 = 5 + ob1 ⇒ ~~& AB(5). operator+ (o2);~~

ob2 = operator + (5, o2);

ob[i] __/__/____

```
friend AB operator* (AB x, AB ob)
{   AB t;
    t.a = x.a * ob.a;
    return t;
}
```

09/9/24

⇒ class Array

```
{ int size;
  int *a;
  public:
    Array (int s)
    { size = s;
    }
    void
```

return reference (be ousafe)

```
int &operator[] (int i)
{ if (i >= size)
    return a[0];
  else return a[i];
}
```

ob[2] = 5;
int x = ob[2];

```
Array ob(5);
ob[5] = 20;  ⟶  ob.operator[](5)
```

We can't overload using member funct.

cout, cin → Inbuilt class functions.
↳ No modification allowed.

⇒ class AB

```
{ int a;
  int b;
  public:
    AB (int a, int b)
    { this->a = a;
      this->b = b;
    }
```

```
int main()
{   AB ob(1,2);
    cout << ob;
```

cout << ob;
cout.operator<<(ob);
operator<<(cout, ob);

__/__/____

=> We have to use friend func^n.

```
friend void operator<< (ostream &os, AB, ob)
  {   os << ob.a;
      os << ob.b;
  }
```

It never changes inbuilt class ostream.

```
AB ob1(1,2), ob2(5,6)
if;   cout << ob1 << ob2;
```

So, we remove void,

Cascading functions.
Can't return VOID,
must return reference.

Works everytime

```
friend ostream& operator << (ostream &os, AB ob)
{   os << ob.a;
    os << ob.b;
}
```

=>

```
class AB
{ int a;
  int b;
public:
  friend void operator >> (istream &is, AB &ob)
  { cout << "Enter values:";
    is >> ob.a;
    is >> ob.b;
  }
```

istream &

reference need.

```
int main()
{ AB ob;
  cin >> ob;
  cin.operator >> (ob);
  operator >> (cin, ob);
```

__/__/____

class AB
{ float a;
public:
AB(float x)
{ a = x;
}

AB ob = 1.5f;
ob.show();

UDT → BDT

```
int main() {
AB ob(5);
int x = ob;   ⟶   x = int(ob);
```

```
class AB
{ int a;
public:
AB ( int x)
{ a = x;
}
operator int( )
{ return a;
}
```

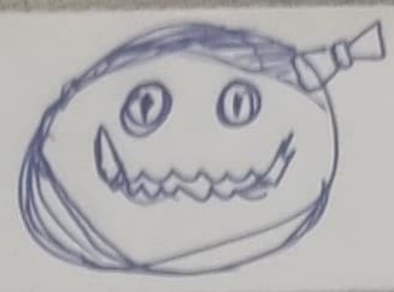
UDT → UDT

```
class S
{ int s;
public:
S(int a)
{ s = a;
}
int get_s()
{ return s;
}
```

```
class D
{ int d;
public:
D(int b)
{ d = b;
}
D ( S ob)
{ d = ob.get_s();
}
```

```
int main ()
{ S obs(5);
D obd;
obd = obs;
```

obd = D(obs)



MR POOKIE.

__/__/____

for multiple inputs in same cin. (CASCADING)

friend istream & → ALWAYS WORKS

Overloading

NEW ⇒

```
class A
{ int a;
  public:
    A(int x)
    { a=x;
    }
    void disp()
    { cout<< "a = "<<a;
    }
int main()
{ A *ptr = new A(5);
  ptr → disp();
  delete ptr;
```

```
void * operator new(size_t sz)
{ void * temp = malloc(sz);
  for(int i=0; i<sz; i++)
  { ((char*) temp)[i] = 0;
  }
}
void operator delete(void *ptr)
{ if(ptr)
    free(ptr);
```

⇒ Conversion Operator

① BDT → UDT ② UDT → BDT ③ UDT → UDT

⇒ BDT → UDT

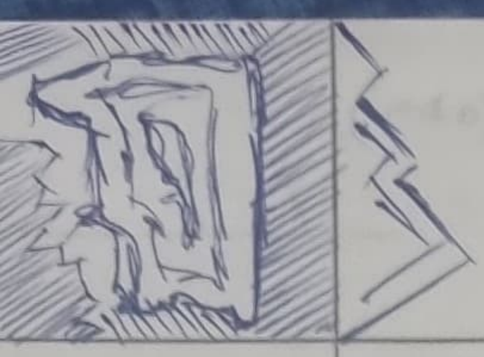
```
class AB
{ int a;
  public:
    AB(int x)
    { a=x;
    }
};
```

Parameterised constr.

```
int main()
{ AB ob = 5;
  ob.show();
}
```

AB ob(5);

__/__/____



If, obs = obd ——> obs = s(obd).

```
s( D ob)
{   s = ob.get_d();
}
```

✗ → Wont work as class D is defined after class S.

Inside class S

```
S (D ob) ;
```

After class D

```
S :: S (D ob)
{   s = ob.get_d();
}
```

✓

Another Way → Overloading Assignment Operator.

obs. operator = (obd) ; obs = obd.

In class S

```
void operator = (D) ;
```

After D

```
void S :: operator= (D ob)
```

__/__/____

```
class A                                    int main()
{ int a;                                   { A ob1(5), ob2;
public:                                      ob2 = ob1++;
  A(int x)
  { a = x;
  }
                                  DUMMY
  A operator++ ((int))
  { return A(a++);
  }
```

⇒ ob2 = 15 - ob1 ⟶ operator-(15, ob1);

```
friend A operator-(int x, A ob)
{   A t;                          (OR)
    t.a = x - ob.a;
    return t;
}
```

ob2 = 15 — (int)ob1

```
operator int()
{ return a;
}
```

obb = oba → (obb) = B(aba)

__/__/____

```
class A                  class B                  int main()
{ int a;                 { int b;                 { A oba(5);
                                                    B obb;
public:                  public:                   obb = oba;
  A(int x)                 B(A ob)                 obb = B(oba); ①
  { a = x;                 { b = ob.get_a();         obb.show();
  }                        }                       obb.operator=(oba) ②
int get_a()                                        }
  { return a;
  }
};
```

① ② .

⇒

```
class B;

class A
{
public:
  operator B();
};

class B
{
};

A::operator B()
{ return B(a);
}
```

② = & Op. Overloading.

```
class B;
class A
{
  void operator=(B
```

A ob1(5), ob2(6), ob3;

ob3 = ob1(100);

ob2 = ob3; → Compiler provides Default Copy Constructor.

```
class Stack {
    int *arr;
    int size;
    int top;
    public:
        Stack(int s) {
            size = s;
            top = -1;
            arr = new int[size];

void push(int e) {
    if (top == size-1)
        throw overflow();
    else {
            top++;
            arr[top] = e;
}
int pop() {
    if (top == -1)
        throw underflow();
    else {
        int val = arr[top]
        top--;
        return val;
    } };
```

__/__/____

```
int main ()
{ A ob1(5), ob2;
  ob2 = (ob1++) + 2;
  cout << ob1;
  cout << ob2;
}
```

WILL WORK AS INT TYPECASTING IS OCCURING

```
operator int ()
{ return a;
}

A operator ++ (int)
{ A t = *this;
  (this -> a) ++;
  return t;
}
```

ob3 = ob1 * (10 + ob2)

⇓

ob3 = (int)(ob1) * (10 + (int) ob2);

⇒ ob3 = (++ob1) * ob2;

⇒ ob3 = ob2 (ob1);

↳ ob3 = ob2.operator()(ob1)

⇒ ob3 = (ob2 += ob1);

```
A operator += (A x)
{   a += x.a;
    return a;
}
```