Operator Overloading - 2

→ Lets try overload some other operators as well "++"

++ → Pre- increment
↳ Post- increment

— Pre- increment

```
int i = 5;
++i;
cout << i; →6
```

i: 6

Exactly, this way we need to overload for our fraction class as well.

f1: N=10, D=2

++f1

$\frac{10}{2} + 1 = \frac{10+2}{2} = \frac{N+D}{D}$

→ $f_3 = f_1 \oplus f_2$

↖ binary operators, operators which need two operands (f_1 & f_2) to work

in such case,

$f_1 + f_2$

f_1 ↑ this, f_2 ↑ argument

→ ++f1;

↖ unary operators, operators which need one operand

++f1;

↑ this

Lets Code it ⟶

```
Void operator++() {
    numerator = numerator + Denominator;
    Simplify();
}
int main() {
    Fraction f1(10,2);
    ++f1;
    f1.Print();
}
```

$\left(\frac{N+D}{D}\right)$

"This Code Works fine"

Now, `int i = 5;`
`j = ++i;`
`Cout << i << " " << j;`
6 6

This must be Possible in fraction as well

`Fraction f2 = ++f1;`
~~`Print(f2)`~~
`f2.Print();`

but, this gives out Error — "no viable Conversion from 'void' to 'fraction'"

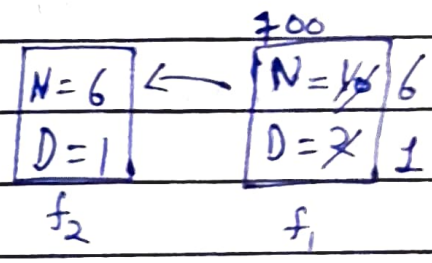
— we are getting Error as our "++" operator is 'void' type, & Since this function is not Returning anything How Can we give its value to "f2".

⟶ Now the Question is what to Return?

Lets See —

```
int i = 5;
int j = ++i;
```

we are Returning 'i'.



Return this; ⊗
it will Return address '700'

Return *this; ✓
Value of this ⟶ f1

⟶ Now our Code is —

```
Fraction operator++() {
        no—— = num — + Den——;
        Simplify();

        Return * this;
    }
int main(){
    Fraction f1(10,2);          output :- 6/1
    Fraction f2 = ++f1;                   6/1     "works fine"
    f1.Print();
    f2.Print();
}
```

⟶ but, Still Some issues Lets See into this

If we do,

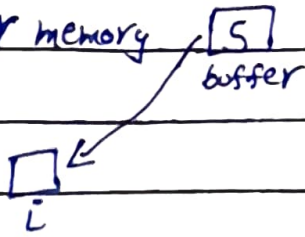
```
int main() {
    Fraction f1(10,2);
        Fraction f2( = ++(++f1);
    f1.Print();
    f2.Print();
}
```

Expected output :- 7/1	original output :- 6/1
7/1	7/1

⟹ Lets See why so,

→ int i = 5;

Internally in System,
this '5' is saved in buffer memory
then 'i' is created
& then value of buffer got
Copied into 'i'.



→ Now In function

```
int fun(int i) {
    i++;
    Return i;
}
main() {
    cout << fun(5);
}
```

i = 5 6

& we Returned 'i' as '6' to the main

Now, If we do

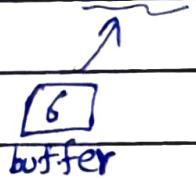
int j = fun(5);

then we can say, 'j' mai '6' chale gaya then we are Printing 'j'

but, what we are doing is

cout << fun(5);

i.e, bina Receive kare directly Print kar Rahe hai



Jo ho Raha ye hai ki,

Jo '6' Return hua - Vo temperatory buffer memory mai Receive hua, & then from buffer only we Print out the value as '6'.

→ Now in our Case Lets see →

f_1 [N = 16 | 6
D = 2 | 1
700

$f_2 = ++(++f_1)$

first $(++f_1)$ is called & we changed in our 'f_1' (6, 1) & we Return *this (Value of 700) or 'f_1'

but, now this Return goes into buffer memory 'x'.

[N = 6
D = 1
x (800)

Now, this = 800

Now, when we Execute ++(++f1)
this '++' got called on 'x'
& our 'x' become

N = 7
D = 1

x 800

& then we call Return *this ⇒ *800 or value of 800
i.e, from buffer we Return our
output as (7,1)
but still in our f1, it is (6,1) as changes
are in buffer only.

⇒ Now, Lets Code Such, to avoid this ——

→ what we want is, System do not create 'buffer' & Pass
f1 only for more changes
i.e, Simple Solution we have to use "Pass by Reference".
so that "buffer will ultimately Point our f1".

⇒ final Code —

```
Fraction & operator++() {
    nu__ = num__ + Den__ ;
    SimPlify();

    Return *this;
}
```

output :- 7/1
7/1

"works fine"