## **User-defined Data Types**

We can also set our own preferences for the input and output operators.

If you overload the input and output operators, your data type behaves like a built-in data type.

```
friend std::istream& operator>> (std::istream& in, Fraction& frac);
friend std::ostream& operator<< (std::ostream& out, const Fraction& frac);</pre>
```

For overloading the input and output operators you have to keep a few rules in mind:

- To support the chaining of input and output operations you have to get and return the input and output streams by non-constant reference.
- To access the private members of the class, the input and output operators have to be *friends* of your data type.
- The input operator >> takes its data type as a non-constant reference.
- The output operator << takes its data type as a constant reference.

```
#include <iostream>

class Fraction{
public:
    Fraction(int num=0, int denom=0):numerator(num), denominator(denom){}

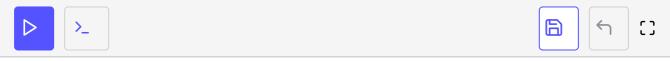
    friend std::istream& operator>> (std::istream& in, Fraction &frac);
    friend std::ostream& operator<< (std::ostream& out, const Fraction& frac);

private:
    int numerator;
    int denominator;
};

std::istream& operator>> (std::istream& in, Fraction& frac){

    in >> frac.numerator;
    in >> frac.denominator;
}
```

```
return in;
}
std::ostream& operator<< (std::ostream& out, const Fraction& frac){</pre>
    out << frac.numerator << "/" << frac.denominator;</pre>
   return out;
}
int main(){
  std::cout << std::endl;</pre>
  Fraction frac(3, 4);
  Fraction frac2(7, 8);
  std::cout << "frac(3, 4): " << frac << std::endl;</pre>
  std::cout << std::endl;</pre>
  std::cout << "Enter two natural numbers for a Fraction: " << std::endl;</pre>
  Fraction fracDef;
  std::cin >> fracDef;
  std::cout << "fracDef: " << fracDef << std::endl;</pre>
  std::cout << std::endl;</pre>
}
```



Overloading input and output operator