User-Defined Data Types

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

If we overload the input and output operators, our 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);
```

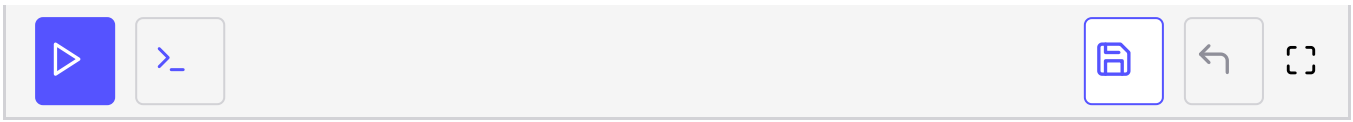
For overloading the input and output operators, we have to keep a few rules in mind:

- To support the chaining of input and output operations, we 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 our 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.

> We first need to enter the values which are spaced apart using the **STDIN** button, then click the **RUN** button. For example, `5 9`

```
#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){

    out << frac.numerator << "/" << frac.denominator;
    return out;
}

int main(){

  std::cout << std::endl;

  Fraction frac(3, 4);
  Fraction frac2(7, 8);
  std::cout << "frac(3, 4): " <<  frac << std::endl;
  std::cout << "frac(7, 8): " << frac2 << std::endl;

  std::cout << std::endl;

  std::cout << "Enter two natural numbers for a Fraction" << std::endl;
  Fraction fracDef;
  std::cin >> fracDef;
  std::cout << "fracDef: " << fracDef << std::endl;

  std::cout << std::endl;

}
```



Overloading input and output operator

In the next lesson, we'll discuss the structure of input and output streams.