Date:

ITEM 21: Don't try to return a reference when you must return an object.

```
class Rational {
    public:
    Rational (int numerator = 0,
              int denominator = 1);
    ...
    private:
    int n, d;

    friend const Rational
    operator* (const Rational& lhs,
               const Rational& rhs);
};
```

⇒ This version of operator* is returning its result by value.

To return objects that are to be multiplied together.

```
Rational a(1, 2);
Rational b(3, 5);
Rational c = a * b; // c should be 3/10
```

Date:.....

its not possible to expect that there already exist rational number with value 3/10

that means, we have to create a new object.

→ It can be created in two ways →
on the stack
on the heap.

⇒ Creatn on Stack → define local variable.

```
const Rational & operator * (const Rational& lhs,
                             const Rational & rhs)
{ Rational result (lhs.n * rhs.n,
                   lhs.d * rhs.d);
  return result;
}
```

⇒ we can reject is approach since we had to avoid ctor call.

Problem with above program is that reference of result is returned from functn & result is local object & gets destroyed when functn ~~exists~~ exits.

Date:.....

=> constructing object on heaps & returning reference to it.

```
const Rational & operator*(const Rational &lhs,
                           const Rational &rhs)
{
    Rational *result = new Rational(lhs.n * rhs.n,
                                    lhs.d * rhs.d);
    return *result;
}
```

=> Still have to call dtor & who will delete the object conjured by new?

```
Rational w, x, y, z;
w = x * y * z;
// same as operator*(operator*(x, y), z)
```

↓

Results in resource leak since we dnt have ptrs hidden behind the references being returned from call to operator*.

=> Another way → static object.

Date:.....

```
const Rational & operator * (const Rational &lhs,
                             const Rational &rhs)
{
    static Rational result;
    result = ...;
    return result;
}
```

Problem ⇒ Consider below client code.

```
bool operator == (const Rational &lhs,
                  const Rational &rhs);
Rational a, b, c, d;
if ((a*b) == (c*d)) // always results
                    // in true.
{ }
else { }
```

Above statement is same as.

```
if (operator == (operator * (a, b),
                 operator * (c, d)))
```

Date:.....

When operator == is called, there are 2 active calls to operator * i.e (a,b) & (c,d).

Thus operator == asked to compare value of static Rational object inside operator * with static Rational object of operator *

No surprise, both are always equal.

=> Now, we can think of static array.

Problem → we don't know the size of n.

=> Right way.

Write a functn that must return a new object is to make that functn return a new object.

```
inline const Rational operator *(const Rational &lhs,
                                 const Rational &rhs)
{
    return Rational(lhs.n * rhs.n,
                    lhs.d * rhs.d);
}
```