Lab 21
class ClassName {

private:
...private data members (variables) and functions...
...anything here is only accessible within the class...

public:
...public data members (variables) and functions...
...anything here is accessible anywhere...
};
class Rat {
private:
int n, d; //These are data members. Rational numbers have a numerator and denominator, so we need two data members here.
public:
//Constructors help us initialize an object’s data members.
Rat() {...}
Rat(int i) {...}
Rat(int i, int j) {...}
//Getter functions help us get the data members’ values.
int getN() {...}
int getD() {...}
//Setter functions help us set the data members’ values.
void setN(int i) {...}
void setD(int i) {...}
//We can define an operator for our Rat objects!
Rat operator+(Rat r) {...}
//These functions are going to defined outside of the Rat class – the friend keyword gives these functions access to Rat’s private data members and functions from outside the Rat class!
friend ostream& operator<<(ostream& os, Rat r);
friend istream& operator>>(istream& is, Rat& r);
};
In main()

Rat r1, r2(5), r3(2, 3); // r1, r2, and r3 are Rat objects

class Rat {
private:
    int n, d; // n = 0, d = 1
public:
    Rat() {
        n = 0;
        d = 1;
    } // default constructor
    Rat(int i) {...}
    Rat(int i, int j) {...}
    ...
};

class Rat {
private:
    int n, d; // n = 5, d = 1
public:
    Rat() {...}
    Rat(int i) {
        n = i;
        d = 1;
    } // 1-argument constructor
    Rat(int i, int j) {...}
    ...
};

class Rat {
private:
    int n, d; // n = 2, d = 3
public:
    Rat() {...}
    Rat(int i) {...}
    Rat(int i, int j) {
        n = i;
        d = j;
    } // 2-argument constructor
    ...
};
A solution which allows us to work with data members is to create public getter and setter member functions. Not all classes should have these – when writing a class, you must decide whether it makes sense to have these functions or not.
Accessors - get and set

```cpp
public:
    int getN() { //get numerator
        return n;
    }
    int getD() { //get denominator
        return d;
    }

class Rat {
    private:
        int n;
        int d;
    public:
        ...;
};

public:
    void setN(int i) {
        n = i;
    }
    void setD(int i) {
        d = i;
    }

int main() {
    Rat r;
    r.setN(52); //calling setN function of Rat object r
    cout << r.getD(); //calling getD function of Rat object r
}
```

Notice how the return types match the data-type of the variable being returned!
Accessors - get and set

If the data members were public, here’s what can happen:

```c
int main() {
    Rat r(3, 4);
    r.d = 0; // we do not want a denominator of 0!
}
```

By keeping the data members private, and only allowing access through member functions, we can guard against bad values being assigned to the data members:

```c
void setD(int i) {
    if (i == 0)
        ...reject i somehow...
    else
        d = i;
}
```
**The sum of two Rat objects is a Rat!**

<table>
<thead>
<tr>
<th>( t.n )</th>
<th>( t.d )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n<em>r.d + d</em>r.n )</td>
<td>( d*r.d )</td>
</tr>
</tbody>
</table>

```cpp
Rat operator+(Rat r) {
    Rat t;
    t.n = n*r.d + d*r.n;
    t.d = d*r.d;
    return t; // t is a Rat
}
```

Rat \( r1(4) \), \( r2(3, 2) \);
Rat \( r3 = r1 + r2 \);
Rat \( r3 = r1 + r2 \);

Direct access to the \( n \) and \( d \) data members of Rat objects \( r \) and \( t \) are possible because the function `operator+` is within the `Rat` class!

The compiler sees this.

Rat `operator+(Rat r) {...}`
class Rat {
private:
int n, d; //Can be accessed by not only functions and constructors within the Rat class, but also by friend functions.
public:
...
friend ostream& operator<<(ostream& os, Rat r);
friend istream& operator>>(istream& is, Rat& r);
};
The idea above also applies to other C++ stream objects, e.g. ofstream, ostringstream, etc.

```
int main() {
    Rat r(4, 3);
    cout « "Rat object: " « r;
    cout.object«("Rat object: ");
    cout.object«(r);
    ostream& operator<<(ostream& os, Rat r) {...}
```
operator>>

```cpp
istream& operator>>(istream& is, Rat& r) {
    is >> r.n >> r.d;
    return is;
}
```

- Streams are not copyable, so we must return by reference.
- The `>>` operator works from left to right.
- C++ has the `>>` operator defined for ints.
- A Rat object is being passed by reference because we want to make changes to its data members.