Logical operators

```
/*

        Topics:
            . Logical operators:
                .#1: Logical operators
                    . We will use the Point class to demonstrate the logical operators.
                    . These will be shown as:
                        . member functions
                        . non-member functions

                .#2: Logical operators with implicit conversions
                    . Laying ground for ⟺
                    . Showing the worst case scenario where you
                      have to implement A LOT of operators.
                    . The number class prevents this by using implicit conversions,
                      because of the explicit constructor.
                    . If we want to support logical comparisons between Number and ints,
                      things like :
                          . 15 < n2
                          . n1 < 25
                          . n1 < n2,
                      we have to implement three versions of the operator<.
                    . If we extend this to all 6 operators, we end up with 18 operators.
                    . This is a lot of boilerplate code.

*/
```

Logical operators

```cpp
Point point1(10.0, 10.0);
Point point2(20.0, 20.0);

std::cout << "point1: " << point1 << "\n";
std::cout << "point2: " << point2 << "\n";


fmt::println("point1 > point2: {}", (point1 > point2));
fmt::println("point1 < point2: {}", (point1 < point2));
fmt::println("point1 ⩾ point2: {}", (point1 ⩾ point2));
fmt::println("point1 ⩽ point2: {}", (point1 ⩽ point2));
fmt::println("point1 == point2: {}", (point1 == point2));
fmt::println("point1 ≠ point2: {}", (point1 ≠ point2));

//Implicit conversions
Number n1(10);
Number n2(20);

fmt::println("n1 < n2: {}", (n1 < n2));
fmt::println("15 < n2: {}", (15 < n2));
fmt::println("n1 < 25: {}", (n1 < 5));
```

Logical operators

```
export class Point
{
  friend std::ostream &operator<<(std::ostream &out, const Point &p);
  friend bool operator>(const Point &left, const Point &right);
  friend bool operator⩾(const Point &left, const Point &right);
  friend bool operator==(const Point &left, const Point &right);
  friend bool operator≠(const Point &left, const Point &right);
  friend bool operator<(const Point &left, const Point &right);
  friend bool operator⩽(const Point &left, const Point &right);

public:
  Point() = default;
  Point(double x, double y) : m_x(x), m_y(y) {}
  ~Point() = default;
  // Operators: can also do them as members.
  /*
  bool operator> (const Point& other) const;
  bool operator< (const Point& other) const;
  bool operator⩾(const Point& other) const;
  bool operator⩽(const Point& other) const;
  bool operator==(const Point& other) const;
  bool operator≠(const Point& other) const;
  */
  double length() const;// Function to calculate distance from the point(0,0)
private:
  double m_x{};
  double m_y{};
};
```

Logical operators: A LOT of them!

```
// Comparison operators
friend bool operator<(const Number &left_operand, const Number &right_operand);
friend bool operator<(int left_operand, const Number &right_operand);
friend bool operator<(const Number &left_operand, int right_operand);

friend bool operator==(const Number &left_operand, const Number &right_operand);
friend bool operator==(int left_operand, const Number &right_operand);
friend bool operator==(const Number &left_operand, int right_operand);

friend bool operator>(const Number &left_operand, const Number &right_operand);
friend bool operator>(int left_operand, const Number &right_operand);
friend bool operator>(const Number &left_operand, int right_operand);

friend bool operator⩾(const Number &left_operand, const Number &right_operand);
friend bool operator⩾(int left_operand, const Number &right_operand);
friend bool operator⩾(const Number &left_operand, int right_operand);

friend bool operator⩽(const Number &left_operand, const Number &right_operand);
friend bool operator⩽(int left_operand, const Number &right_operand);
friend bool operator⩽(const Number &left_operand, int right_operand);

friend bool operator≠(const Number &left_operand, const Number &right_operand);
friend bool operator≠(int left_operand, const Number &right_operand);
friend bool operator≠(const Number &left_operand, int right_operand);
```