Intro C++20 “spaceship” operator <=>

Diego Rodriguez-Losada
@diegorlosada

C++ Meetup, Madrid, 20-Feb-2020

C++17?

```
struct MyInt {
  int value;
  constexpr MyInt(int value): value{value} { }
  bool operator==(const MyInt& rhs) const { return value == rhs.value; }
  bool operator!=(const MyInt& rhs) const { return !(*this == rhs);    }
  bool operator<(const MyInt& rhs)  const { return value < rhs.value;  }
  bool operator<=(const MyInt& rhs) const { return !(rhs < *this);     }
  bool operator>(const MyInt& rhs)  const { return rhs < *this;        }
  bool operator>=(const MyInt& rhs) const { return !(*this < rhs);     }
};
```

default <=>

```cpp
#include <compare>

struct MyInt {
    int value;
    auto operator<=>(const MyInt&) const = default;
};

int main() {
    MyInt a{ 1 }, b{ 2 };
    std::cout << (a < b) << "\n"; // 1
    std::cout << (a > b) << "\n"; // 0
    std::cout << (a == b) << "\n"; // 0
}
```

rewritten expressions

```
#include <compare>

struct MyInt {
    int value;
    auto operator<=>(const MyInt&) const = default;
};

int main() {
    MyInt a{ 1 }, b{ 2 };
    std::cout << ((a <=> b) < 0) << "\n"; // 1
    std::cout << ((a <=> b) > 0) << "\n"; // 0
    std::cout << ((a <=> b) == 0) << "\n"; // 0
}
```

<=> return type (default)

```cpp
#include <compare>

struct MyInt {
    int value;
    auto operator<=>(const MyInt&) const = default;
};

int main() {
    MyInt a{ 1 }, b{ 2 };
    std::cout << ((a <=> b) < 0) << "\n"; // 1
    std::cout << ((a <=> b) > 0) << "\n"; // 0
    std::cout << ((a <=> b) == 0) << "\n"; // 0
    std::cout << typeid(a <=> b).name() << "\n"; //class std::strong_ordering
    //std::strong_ordering::less < 0
    //std::strong_ordering::greater > 0
}
```

Synthesized expressions

```
struct MyInt {
    int value;
    constexpr MyInt(int value) : value{ value } { }
    auto operator<=>(const MyInt&) const = default;
};

int main() {
    MyInt a{ 1 }, b{ 2 };

    std::cout << (4 > a) << "\n"; // 1
    // Equivalent
    std::cout << (0 > (a <=> 4)) << "\n"; // 1
}
```

Any type!

```
struct Age {
    int value;
    auto operator<=>(const Age&) const = default;
};

struct Person {
    Age age;
    int height;
    auto operator<=>(const Person&) const = default;
};

int main() {
    Person diego{ 34, 183 };
    Person dani{ 27, 184 };
    Person juan{ 34, 207 };
    std::cout << (diego > dani) << "\n";  // 1
    std::cout << (diego > juan) << "\n";  // 0
}
```

Custom implementation

```
struct Age {
    int value;
    auto operator<=>(const Age&) const = default;
};
struct Person {
    Age age;
    int height;
    auto operator<=>(const Person& p) const {
        if (auto c = height <=> p.height; c != 0) return c;
        return age <=> p.age;
    };
};
int main() {
    Person diego{ 34, 183 };
    Person dani{ 27, 184 };
    Person juan{ 34, 207 };
    std::cout << (diego > dani) << "\n";  // 0
    std::cout << (diego > juan) << "\n";  // 0
}
```