Ordering examples

```
/*
    . Topics:

        . Exploring weak and partial ordering:
            .#1: weak ordering example1
            .#2: weak ordering example2
            .#3: partial ordering example.

*/
```

Weak ordering example #1

```cpp
//#1: Weak ordering example1
class ComparableString
{
public:
ComparableString(const std::string &str) : m_str{ str } {}

std::weak_ordering operator<=>(const ComparableString &right_side) const
{

    if (m_str.size() == right_side.m_str.size()) {
    return std::weak_ordering::equivalent;
    } else if (m_str.size() > right_side.m_str.size()) {
    return std::weak_ordering::greater;
    } else {
    return std::weak_ordering::less;
    }
}

bool operator==(const ComparableString &right_side) const { return (m_str.size() == right_side.m_str.size()); }

private:
std::string m_str;
};
```

Weak ordering example #1

```
//#1: Weak ordering example1
export void weak_ordering_ex1(){
    ComparableString cmp_str1("Fog");
    ComparableString cmp_str2("Dogy");

    fmt::println("cmp_str1 > cmp_str2 : {}", (cmp_str1 > cmp_str2));
    fmt::println("cmp_str1 < cmp_str2 : {}", (cmp_str1 < cmp_str2));

    fmt::println("cmp_str1 == cmp_str2 : {}", (cmp_str1 == cmp_str2));
}
```

Weak ordering example #2

```cpp
// Weak ordering example 2
std::weak_ordering case_insensitive_compare(const char *str1, const char *str2) {
    // Turn them all to uppercase
    std::string string1(str1);
    std::string string2(str2);

    for (auto &c : string1) {
        c = toupper(c);
    }

    for (auto &c : string2) {
        c = toupper(c);
    }

    int cmp = string1.compare(string2);
    if (cmp > 0)
        return std::weak_ordering::greater;
    else if (cmp == 0)
        return std::weak_ordering::equivalent;
    else
        return std::weak_ordering::less;
}
```

Weak ordering example #2

```cpp
// Weak ordering example 2
class CaseInsensitiveString {
public:
    CaseInsensitiveString(const std::string &str) : s(str) {}

    std::weak_ordering operator<=>(const CaseInsensitiveString &b) const {
        return case_insensitive_compare(s.c_str(), b.s.c_str());
    }

    std::weak_ordering operator<=>(const char *b) const {
        return case_insensitive_compare(s.c_str(), b);
    }

    bool operator==(const CaseInsensitiveString &right_operand) const {
        return (case_insensitive_compare(s.c_str(), right_operand.s.c_str()) == std::weak_ordering::equivalent);
    }

private:
    std::string s;
};
```

Weak ordering example #2

```
// Weak ordering example 2
export void weak_ordering_ex2() {
    CaseInsensitiveString ci_str1("Hello");
    CaseInsensitiveString ci_str2("ZELLO");

    fmt::println("ci_str1 ⩽ ci_str2 : {}", (ci_str1 ⩽ ci_str2));
    fmt::println("ci_str1 ⩽ Kello : {}", (ci_str1 ⩽ "Kello"));
    fmt::println("Kello ⩽ ci_str2 : {}", ("Kello" ⩽ ci_str2));

    // You need to put in a == operator. Compiler won't generate one for you.
    fmt::println("ci_str1 == ci_str2 : {}", (ci_str1 == ci_str2));
}
```

Partial ordering #1

```cpp
// Valid comparable points are within the [100,100] bounds
class Point
{
public:
Point(int x, int y) : m_x{ x }, m_y{ y } {}
bool operator==(const Point &right) const {
    return (*this <=> right) == std::partial_ordering::equivalent;
}
std::partial_ordering operator<=>(const Point &right) const
{
    if (is_within_bounds(*this) && is_within_bounds(right)) {
    if (length() > right.length())
        return std::partial_ordering::greater;
    else if (length() < right.length())
        return std::partial_ordering::less;
    else
        return std::partial_ordering::equivalent;
    }
    return std::partial_ordering::unordered;
}

private:
bool is_within_bounds(const Point &p) const
{
    if ((std::abs(p.m_x) < 100) && (std::abs(p.m_y) < 100)) return true;
    return false;
}
double length() const { return sqrt(pow(m_x - 0, 2) + pow(m_y - 0, 2) * 1.0); }
int m_x{};
int m_y{};
};
```

Partial ordering #1

```
export void partial_ordering_example(){

    Point p1(110, 110);
    Point p2(20, 20);

    auto result1 = (p1 > p2);
    fmt::println("p1 > p2 : {}", result1);

    auto result2 = (p1 ⩾ p2);
    fmt::println("p1 ⩾ p2 : {}", result2);

    auto result3 = (p1 == p2);
    fmt::println("p1 == p2 : {}", result3);

    auto result4 = (p1 ≠ p2);
    fmt::println("p1 ≠ p2 : {}", result4);

    auto result5 = (p1 < p2);
    fmt::println("p1 < p2 : {}", result5);

    auto result6 = (p1 ⩽ p2);
    fmt::println("p1 ⩽ p2 : {}", result6);
}
```