1 Freie Funktionen

Bei der Überladung von binären Operatoren wird die Variante der freien Funktion bevorzugt, da sie die implizite Konvertierung für beide Operanden ermöglicht. In dieser Aufgabe sollen Sie eine Klasse `OneSide` mit `operator==` als Member-Funktion und eine Klasse `TwoSide` mit freier Funktion implementieren.

1.1 Aufgabe

Implementieren Sie beide Klassen mit einem einzelnen **`int`** als Attribut und einem einzelnen Konstruktor, welcher dieses Attribut mit einem beliebigen Wert initialisiert. Damit dieser Konstruktor auch für implizite Konvertierungen von **`int`** zu `OneSide` resp. `TwoSide` funktioniert, dürfen Sie ihn nicht `explicit` markieren, stattdessen aber cpplint mittels `// NOLINT[runtime/explicit]` zufriedenstellen.

Nun können Sie den `operator==` einmal als Member-Methode (mit nur einem Parameter) in `OneSide` implementieren und einen entsprechenden Test schreiben. Sie werden feststellen, dass ein Ausdruck (`a` ist eine `OneSide` Instanz) wie **`if`** `(a == 1)` funktioniert, **`if`** `(1 == a)` aber nicht.

Implementieren Sie den `operator==` nun als freie Funktion (mit zwei Parameter und der `friend`-Markierung) in `TwoSide`. Mit dieser Variante sollten nun beide Variante des oben gezeigten Vergleichs funktionieren.

Versuchen Sie zu erklären, was hier genau passiert und weshalb die Variante mit Member-Methode nicht funktioniert.

2 Relationale Operatoren

Üblicherweise werden wenige bis keine Operatoren von eigenen Klassen definiert. Neben den Assignment und Stream-Operatoren sind aber die relationalen Operatoren häufig sinnvoll. Vorallem wenn die eigene Klasse innerhalb eines Containers wie z.B. `std::set` verwendet werden soll. Wir wollen uns hier somit Operator-Overloading zunutze machen.

Dabei sollte man sich zunutze machen, dass die meisten relationalen Operatoren durch einen der anderen definiert werden kann. Z.B. können Sie `!=` durch die Invertierung vom Resultat von `==` implementieren. Verwenden Sie auch hier freie Funktionen für die Implementierung der Operatoren.

Eine übersicht über die relationalen Operatoren finden z.B. hier: cppreference.com - Comparison operators. Der `operator<=>` wird hier bereits erwähnt, sollten Sie aber nicht verwenden, da der C++20-Standard noch nicht definitiv ist.

2.1 Aufgabe

Implementieren Sie die Klasse `ComparableData` welche ein **`int`** und ein **`double`** Attribut hat. Diese Klasse sollen Sie später in einer sortieren Menge vom Typ `std::set<ComparableData>` verwenden können. Bei der Sortierung soll das **`int`** Attribut höhere Priorität haben. Schreiben Sie zudem ein paar Tests, welche die korrekte Sortier-Reihenfolge überprüfen.