1 Generic Maximum

In den Folien finden Sie die Definition des Funktions-Templates `min()`. Implementieren Sie eine verallgemeinerte Version von `max()`, die überall (Return-Typ und alle Parameter) verschiedene Typen annehmen kann. Für das Testen der Funktionalität können Sie folgenden Code verwenden:

```
1   const int i = 3;
2   const double d = 3.14;
3
4   std::cout << max(2, i) << std::endl;
5   std::cout << max<double>(i, d) << std::endl;
6   std::cout << max(i, d) << std::endl;
```

1.1 Aufgabe

Implementieren Sie die Funktion auf unterschiedliche Arten (mittels Überladung) und analysieren Sie, wann welche Variante aufgerufen wird.

2 Container-Casting

Implementieren Sie ein Klassen-Template `Container`, welches ein einziges Element seines Template-Parameters speichert. Folgender Code soll dann möglich sein:

```
1   Container<int> a(1);
2   std::cout << a << std::endl;                    // 1
3   Container<double> b(3.14);
4   std::cout << b << std::endl;                    // 3.14
5   a = b;
6   std::cout << a << std::endl;                    // 3
7   std::cout << Container<int>(8.91) << std::endl;   // 8
```

2.1 Aufgabe

Implementieren Sie die Klasse. Zudem sollen Sie einen Konvertierungs-Konstruktor und einen Copy-Assignment-Operator implementieren, welche beide ebenfalls Templates sind.

3 Any-Creator

In dieser Aufgabe sollen Sie ein Funktions-Template schreiben, welches einen `std::vector` von `std::any` Objekten erstellt. Folgender Code soll damit möglich sein.

```
1  auto container = CreateAnyVector("Hello", 3.14, 'A', true, 42);
2  for (const auto& any : container) {
3    std::cout << any.type().name() << std::endl;
4  }
```

3.1 Aufgabe

Implementieren Sie die Funktion `CreateAnyVector` mittels Variadic Templates. Damit Sie die Pack-Expansion für einzelne Funktionsaufrufe machen können, bietet sich Rekursion oder eine Fold-Expression an.